Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Urh Svetičič

Sistem za nadzor ribogojnice

diplomsko delo

univerzitetni strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana 2015

Fakulteta za računalništvo in informatiko podpira javno dostopnost znanstvenih, strokovnih in razvojnih rezultatov. Zato priporoča objavo dela pod katero od licenc, ki omogočajo prosto razširjanje diplomskega dela in/ali možnost nadaljne proste uporabe dela. Ena izmed možnosti je izdaja diplomskega dela pod katero od Creative Commons licenc http://creativecommons.si

Morebitno pripadajočo programsko kodo praviloma objavite pod, denimo, licenco *GNU General Public License, različica 3.* Podrobnosti licence so dostopne na spletni strani http://www.gnu.org/licenses/.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Razvijte vgrajen sistem, ki bo omogočal nadzor osnovnih parametrov v ribogojnicah. Sistem naj bo zgrajen na osnovi mikrokrmilnika STM32F407. Vgrajen sistem naj preko mrežne povezave pošilja podatke na računalnik Raspberry Pi 2, kjer se gradi podatkovno bazo. Celoten sistem naj bo mogoče nadzorovati preko spletnega vmesnika.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Urh Svetičič sem avtor diplomskega dela z naslovom:

Sistem za nadzor ribogojnice

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Patricia Bulića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 3. septembra 2015 Podpis avtorja:

Rad bi se zahvalil podjetju Faronika d.o.o., še posebej g. Dušanu Jesenšku, saj so mi z nakupom senzorjev omogočili izdelavo diplomske naloge v takem obsegu.

Poleg tega bi se zahvalil mentorju Patriciu Buliću in asistentu Roku Češnovarju, ker sta mi pomagala pri izdelavi te diplomske naloge.

Na koncu se zahvaljujem še družini in prijateljem, ker so me podpirali in mi pomagali v celotnem času študija.

Diplomsko nalogo posvečam noni Ančki.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
RISC	Reduced Instruction Set Computer	računalnik s skrčenim naborom ukazov
USART	Universal Sychronous Asychronous Reciver Transmiter	univerzalni sinhroni asinhroni sprejemnik in oddajnik
UART	Universal Asychronous Reciver Transmiter	univerzalni asinhroni sprejemnik in oddajnik
CAN	Controller Area Network	krmilnik omrežja
SDIO	Secure Digital Input Output	varni digitalni vhodi izhodi
USB OTG	Universal Serial Bus On-The-Go	vsestransko zaporedno vodilo
GPIO	General purpose input output	splošno namenski vhodi/izhodi
PIN	Personal Identification Number	osebna identifikacijska številka
LSB	Lest significant bit	najmanj pomembni bit
PWM	Pulse Width modulation	modifikacija širine pulza
GND	Ground	ozemljitev
V	Volt	Volt
BPS	bits per second	bitov na sekundo
TCP	Transmission Control Protocol	
UDP	User Datagram Protocol	
ARM	Advanced RISC Machines	
I2C	Inter-Integrated Circuit	
SPI	Serial Port Interface Bus	
I2S	Integrated Interchip Sound	
...	...	...

Povzetek

Namen tega diplomskega dela je razviti sistem, ki bo omogočal nadzor osnovnih parametrov v ribogojnicah. Zato smo naredili vgrajeni sistem, ki je sestavljen iz štirih senzorjev ter ploščice STM32F4 Discovery, ta se nato preko modula ethernet URS-TCP-232-T poveže na Raspberry Pi 2, kjer se gradi podatkovno bazo. Vse skupaj pa se nadzoruje preko spletnega vmesnika.

Diplomska naloga je sestavljena iz dveh delov. Prvi del je namenjen teoretičnemu uvodu, v katerega smo vključili kratek opis delovanja ribogojnice in to, kateri parametri so pomembni in jih je treba nadzorovati. Sledi podroben pregled elementov, ki jih bomo potrebovali za delovanje našega sistema, njihova uporaba ter opis tehnologij in orodij, potrebnih za izdelavo našega sistema. V praktičnem delu najprej opišemo delovanje vgrajenega sistema, katere GPIO pine smo uporabili in kako smo upravljali senzorje s pomočjo FreeRTOS opravil. V nadaljevanju pa se posvetimo programiranju Raspberry Pi strežnika, načrtovanju in kreiranju baz MySql in RRD ter izdelavi spletne strani s pomočjo HTML, CSS in PHP jezika. Strežniku smo dodali še dodatno funkcionalnost opozarjanja z SMS obvestili preko gsm modula Adafruit FONA.

Ključne besede: vgrajeni sistem, ribogojnica, USR-TCP232-T, nadzor, STM32F4 Discovery, Adafruit FONA, DHT22, Maxbotix MB7060, Atlas EZO Dissolved Oxygen kit.

Abstract

The purpose of this bachelor's theses is to develop a system that will enable monitoring over the basic parameters in fish farms. That is why we have made an embedded system which is composed of four sensors and the STM32F4 Discovery board. This board is then connected through Ethernet module to Raspberry Pi 2, where the database is built. All together is monitored through web interface.

The paper is composed of two parts. The first part is intended for a theoretical introduction in which we have included a short description of the fish farm operation, and pointed out which parameters are important and necessary to be monitored. What follows is a detailed overview of the elements needed for our system to work, their use, and description of technologies and tools needed to build our system. In the practical part, we first describe the operation of the embedded system, which GPIO pins we used, and how we managed the sensors with the help of FreeRTOS tasks. Further on, we dedicate ourselves to the programming of the Raspberry Pi server, planning and designing of MySql and RRD bases, and building a website with the help of HTML, CSS and PHP languages. We have added an extra functionality of warning with SMS notifications through GSM module Adafruit FONA.

Keywords: embedded system, fish farm, USR-TCP232-T, monitoring, Adafruit FONA, DHT22, Maxbotix MB7060, Atlas EZO Dissolved Oxygen kit.

Poglavje 1

Uvod

Človek si že od samega začetka poskuša olajšati delo z uporabo različnih naprav. Na začetku so bile to čisto preproste stvari, sestavljene iz parih delov, ki so jih lahko našli v naravi. Z leti pa so le-te postajale vedno bolj sofisticirane. Trenutno je najbolj na udaru računalništvo, ki zaradi hitrega razvoja ponuja sisteme, ki so majhni, zmogljivi in predvsem cenovno ugodni. V vseh gospodarskih panogah se torej poskuša avtomatizirati procese, pri čemer se pridobi na času in zmanjša človeško prisotnost. Večina računalniško vodenih sistemov se nahaja v industriji, na drugi strani pa imamo kmetijstvo, ki je s stališča računalništva še dokaj nedotaknjeno področje.

Ribogojstvo je ena izmed najhitreje rastočih živilskih panog na svetu in že predstavlja skoraj polovico vseh rib za prehrano ljudi [2]. Iz navedenega razloga so potrebe po avtomatizaciji velike. Glede na to, da večina ribogojnic deluje po istem principu, bo mogoče naš sistem z minimalnimi priredbami uporabljati tudi v drugih ribogojnicah.

V prvem delu diplomske naloge smo opisali delovanje ribogojnice, ki nam da vpogled v to, kateri parametri so pomembni in jih je mogoče meriti in nadzorovati na daljavo. Nato sledi pregled elementov, orodij in protokolov, ki smo jih potrebovali za izdelavo vgrajenega sistema, namenjenega nadzorovanju stanja v ribogojnici. "Vgrajeni sistem je elektronski sistem, ki vključuje enega ali več mikrokrmilnikov in je zgrajen tako, da izvaja določeno namen-

sko nalogo"[3]. Za tem sledi še opis Raspberry Pi 2, ki smo ga uporabili kot strežnik ter opis ostalih uporabljenih programov za delovanje strežnika.

V drugem delu pa je opisana natančna sestava vezja ter načrtovanje programa, ki ga bo poganjala razvojna ploščica STM32F4 Discovery. Za hkratno delovanje vseh senzorjev in modulov se je s pomočjo operacijskega sistema FreeRTOS ustvarilo pet opravil, ki so bila zadolžena za komunikacijo s senzorji. Preko istega čipa se je podatke, pridobljene iz štirih senzorjev, pošiljalo do relacijske baze MySql in baze s krožnim dodeljevanjem RRDtool. V bazo se je shranjevalo podatke o globini in temperaturi vode, vsebnosti kisika v vodi, temperaturi zraka in vlagi. Za prikaz podatkov se je naredilo spletno stran, ki s pomočjo PHP, RRDtool in MySql prikazuje grafe in podatke, zabeležene v bazi. V primeru kritičnih vrednosti parametrov smo strežniku dodali možnost opozarjanja z SMS obvestili. Na koncu smo ta sistem še testirali ter zapisali možne izboljšave ter ugotovitve.

Poglavje 2

Ribogojstvo

Kot smo že v uvodu omenili, je ribogojstvo trenutno ena izmed najbolj rastočih živilskih panog. Vendar ribogojstvo ni namenjeno izključno vzreji rib za prehrano. V splošnem imamo dva namena uporabe ribogojstva. Prvi je namenjen vzreji rib za prehranjevanje, ki ga prakticira večina ribogojnic. Drugi pa je namenjen vzreji rib za vlaganje v odprte vode. Ribogojnica Tolminka (Slika 2.1), na kateri bomo testirali naš sistem, je bila primarno zgrajena z namenom vzreje rib za vlaganje, del rib pa bo namenjen prodaji v kulinarične namene.

Slika 2.1: Ribogojnica Tolminka.

Zaradi prekomernega izlova rib v preteklosti in zaradi vnosa tujerodne potočne postrvi je prišlo do zmanjšanja populacije avtohtone soške postrvi (*lat. Salmo trutta marmoratus*) (Slika 2.2) in lipana (*lat. Thymallus thymallus*) (Slika 2.3) v reki Soči ter njenih pritokih. Zato je lokalna ribiška družina začela z raziskavami in repopulacijskim gojenjem teh dveh vrst. Z delom na tem področju jim je v zadnjih letih uspelo kar precej izboljšati stanje. S tem uspehom so si zadali cilj zmanjšati vlaganje tujerodnih vrst za potrebe športnega ribolova in ga v čim večji meri nadomestiti z vlaganjem avtohtonih rib. Za dosego tega cilja pa se je pojavila potreba po izgradnji večje ribogojnice, ki so jo dokončali v letošnjem letu. Novonastala ribogojnica je namenjena vzreji večjih soških postrvi ter lipanov. V splošnem je vzreja vodnih živali bolj rizična od vzreje kopenskih, saj je potrebna stalna prisotnost vode. V testni ribogojnici pa je bila potreba po nadzornem sistemu še toliko večja, saj sta soška postrv in lipan precej bolj občutljiva kot pa kalifornijska postrv (*lat. Oncorhynchus mykiss*), ki se jo ponavadi najde v ribogojnicah.

Slika 2.2: Soška postrv *lat. Salmo trutta marmoratus*.

Slika 2.3: Lipan *lat. Thymallus thymallus.*

Parametri za nadzor ribogojnice

Najpomembnejša stvar, ki jo potrebujemo za normalno delovanje ribogojnice, je seveda voda. Ribogojnica Tolminka ima na začetku velik zalogovnik vode, ki skrbi, da je v spodnjih bazenih vedno dovolj vode. Zaradi razvojne faze našega sistema smo so odločili, da bomo vse meritve opravljali v zalogovniku. S tem smo znižali ceno našega sistema in poenostavili montažo ter kasnejša popravila.

Vendar za ribe ni pomembna samo količina vode, ampak tudi to, kakšne so njene lastnosti. Soška postrv in lipan sta rečni ribi, ki za vzrejo zahtevata dokaj visoko vsebnost kisika in precej nizke temperature. Za večje soške postrvi se priporoča, da je temperatura vode vedno pod 18 °C, medtem ko je optimalna temperatura nekje med 8 °C in 15 °C. Za vsebnost raztopljenega kisika pa se priporoča, da je nad 5 mg/l. Lipan je še bolj občutljiv kot soška postrv in zahteva, da je temperatura vode pod 13 °C, optimalna temperatura je med 8 °C in 10 °C, vsebnost kisika pa mora biti nad 7 mg/l. Kljub temu da so te vrednosti kar striktne, za ribogojnico niso problematične, saj se oskrbuje z vodo, pridobljeno iz Tolminke, ki ima skozi celo leto primerno temperaturo in visoko vsebnost kisika. Ne glede na to, da imamo stalni pritok vode, smo se

vseeno odločili te parametre nadzorovati in s tem zagotavljati večjo varnost v primeru nepričakovanih dogodkov.

Poleg zgoraj naštetih parametrov bomo merili še temperaturo zraka in vlago, kar bo služilo bolj kot informativni podatek, saj ne vpliva na samo delovanje ribogojnice. Naš sistem bo z interneta pridobival tudi informacijo o pretoku vode v ribogojnico, ki pa ga zaradi tuje implementacije v nadaljevanju ne bomo omenjali.

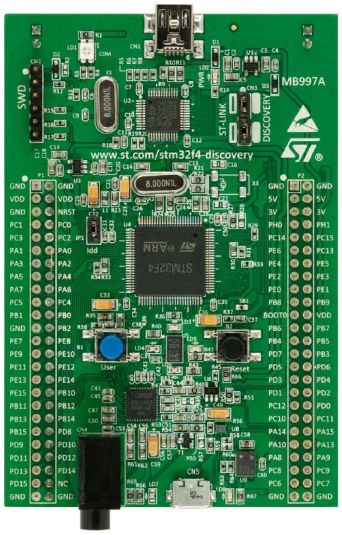
Poglavje 3

Opis tehnologij, orodij in komponent, potrebnih za izdelavo vgrajenega sistema

Za nadziranje stanja na ribogojnici je bilo treba narediti svoj sistem za zajem podatkov. Glede na zahteve ribogojnice smo se odločili meriti gladino vode, vsebnost raztopljenega kisika v vodi, temperaturo vode, vlažnost ter temperaturo zraka. Za zajem in posredovanje podatkov na strežnik smo potrebovali nek mikrokrmilnik. Po posvetu z mentorjem smo se odločili za razvojno ploščico STM32F4 Discovery, saj ponuja velik razpon komunikacijskih protokolov, za procesiranje pa uporablja ARM Cortex, ki je dovolj zmogljiv za naše zahteve.

3.1 Stm32F4 Discovery

Mikrokrmilnik STM32F4 Discovery (Slika 3.1) [6] je del serije STM32F407xx, ki temelji na visoko zmogljivem procerskem jedru ARM®-Cortex®-M4. Procesor Cortex-M4 vsebuje 32-bitno RISC (ang. Reduced Instruction Set Computer) jedro, kar pomeni, da lahko operira nad 32 bitnimi spremenljivkami in naslavlja 2^{32} besed. Zaradi RISC strukture ima omejen nabor ukazov,



Slika 3.1: Mikrokrmilnik STM32F4 Discovery.

kar mu omogoča hitrejše izvajanje, saj lahko vsak ukaz izvede v enem urinem ciklu. Procesor deluje s frekvenco 168 MHz ter vsebuje dodatno enoto za računanje z decimalnimi števili v enojni natančnosti (*ang. floatingpoint unit*). Za delovanje ima na voljo 1 megabajt pomnilnika, na katerega se naloži program, ter do 192 kilobajtov SRAM-a, ki ga uporablja procesor kot predpomnilnik. Za priklop senzorjev imamo na voljo tri 12-bitne analogno-digitalne pretvornike. Zraven je vključenih tudi dvanajst 16-bitnih časovnikov ter dva 32-bitna.

3.1.1 Komunikacijski vmesniki

Za komunikacijo imamo na voljo naslednje komunikacijske vmesnike:

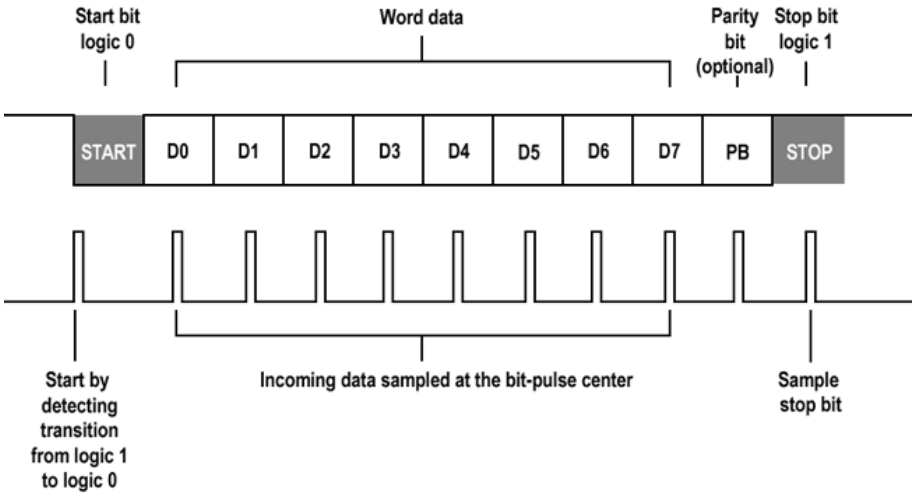
- do tri I2C (*Inter-Integrated Circuit*) povezave
- tri SPI (*Serial Port Interface Bus*) povezave

- dve oboje smerni I2S (*Integrated Interchip Sound*) povezavi za natančen prenos zvoka
- štiri U(S)ART (*Universal Sychronous Asychronous Reciver Transmiter*) povezave
- dve UART (*Universal Asychronous Reciver Transmiter*) povezavi
- dve CAN (*Controller Area Network*) povezavi
- eno SDIO (*Secure Digital Input Output*) povezavo
- en micro USB OTG (*Universal Serial Bus On-The-Go*) in en mini USB OTG s polno zmogljivostjo

Ker večkrat uporabimo protokol UART (Slika 3.2), ga bomo bolj natančno razložili. UART je asinhroni serijski protokol, kar pomeni, da se podatki prenašajo zaporedno in brez dodatne urine sinhronizacije. Zaradi odsotnosti ure je treba napravam, ki komunicirajo preko UART-a, nastaviti hitrost prenosa bitov na sekundo (*ang. baudrate*). Pri tem moramo biti pozorni, da notranji uri naprav ne odstopata za več kot 10 %. Pred začetkom prenosa podatkov pošiljatelj najprej pošlje start bit, s tem da postavi vodilo na logično 0. Po start bitu začne pošiljati 8 bitov podatkov. Pošilja po principu najmanj pomemben bit prvi oziroma LSB first (*ang. lest significant bit first*). Po poslanem enem bajtu se lahko opcijsko pošlje en paritetni bit. Komunikacijo se zaključi s stop bitom, ko pošiljatelj postavi vodilo na logično 1.

3.1.2 Vhodno-izhodne enote

Na razvojnem vezju STM32F4 Discorey imamo na voljo 82 splošno namenskih vhodno izhodnih pinov GPIO (Slika 3.3) (*ang. General purpose Input Output*). Za uporabo določenega pina je treba najprej vklopiti uro na vodilu, nato pa preko štirih 32-bitnih nastavitvenih registrov in dveh podatkovnih registrov nastavimo željeno delovanje pina. Pin lahko konfiguriramo kot vhodni

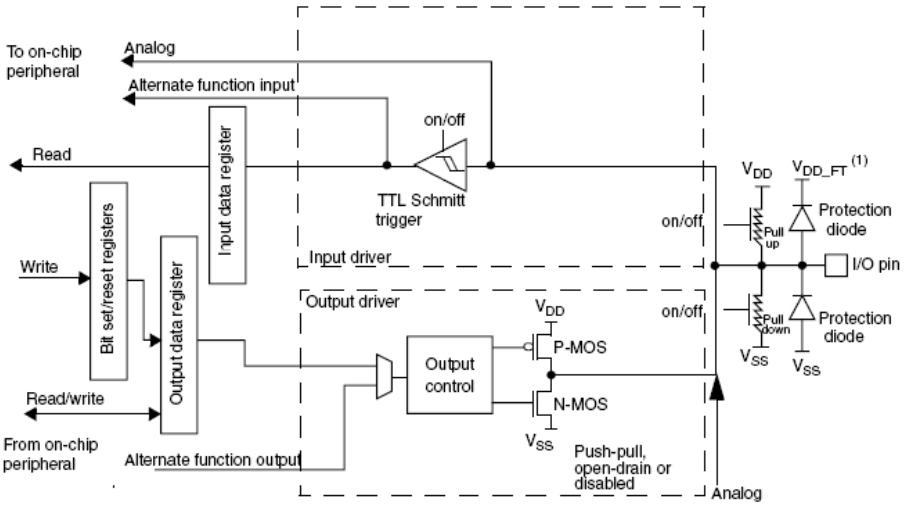


Slika 3.2: Shema UART komunikacije. [5]

oziroma izhodni, poleg tega mu lahko še določimo hitrost, napetostno stanje, PWM (*ang. Pulse Width modulation*), upor ali pa ga vežemo na določen komunikacijski protokol s pomočjo alternativne funkcije.

3.2 FreeRTOS

FreeRTOS je popularen odprtokodni operacijski sistem, ki deluje v realnem času. Namenjen je vgrajenim sistemom in trenutno nudi podporo za 35 različnih mikrokrmilnikov. Zaradi omejene zmogljivosti mikrokrmilnikov je zelo majhen in enostaven, zato za njegovo delovanje potrebujemo samo štiri C datoteke. FreeRTOS za razliko od ostalih večjih operacijskih sistemov ne vključuje gonilnikov, omrežnih povezav, upravljanja s pomnilnikom, ampak samo najosnovnejše elemente, kot je preklapljanje med opravili, ključavnice, vrste, semaforje in programske časovnike. Poleg tega ima še tick-less mode, kar nam zmanjša porabo energije.



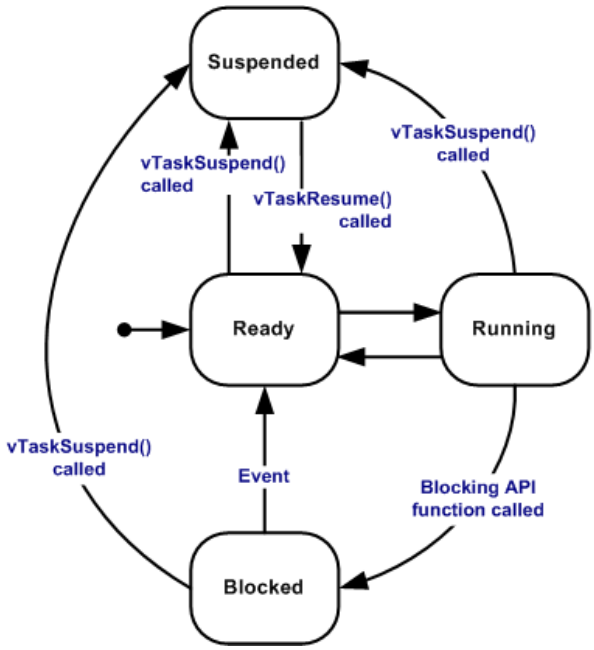
Slika 3.3: Shema vhodno-izhodnih pinov.[7]

3.2.1 Preklaplanje med opravili

Za preklapljanje med opravili uporablja SysTick prekinitev v navezi s PendSv. Ko pride do SysTick prekinitve, ki se zgodi ob točno določenih urinih periodah, se v prekinitvi proži PendSv prekinitev, ki ima nižjo prioriteto. Potrebo po proženju dodatne prekinitve imamo zato, ker bi v nasprotnem primeru prišlo do nepravilnega vračanja iz prekinitev in bi posledično imeli odseke kode, ki se ne bi nikoli več izvedli.

3.2.2 Stanja opravil

FreeRTOS podpira štiri stanja opravil (Slika 3.4). Opravilo je lahko blokirano, pripravljeno, teče ali pa je suspendirano. Pri STM-u je lahko zaradi enojedrnega procesorja v tekočem stanju samo eno opravilo, vsa opravila, ki čakajo na procesorski čas, pa so v pripravljenem stanju. Katero opravilo bo naslednje na vrsti za tekoče stanje, je odvisno od prioritetne vrste, ki jo določimo ob kreiranju opravila. Opravila v blokirajočem stanju čakajo na dogodek, ki jih bo spravil v pripravljeno stanje. Ostane nam še suspendirano stanje, iz katerega je mogoče priti le, če od drugega opravila dobimo klic



Slika 3.4: Shema prehodov med stanji opravil.[9]

vTaskResume in s tem preidemo v pripravljeno stanje.

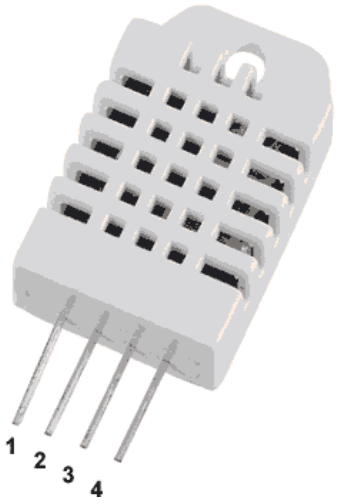
3.3 IAR Embedded Workbench

IAR Embedded Workbench je programsko orodje, namenjeno razvoju vgrajenih sistemov. Podpira večino ARM arhitektur in je namenjeno pisanju v programskih jezik C in C++. Poleg tega je vanj vključeno še zapisovanje programa na pomnilnik, razhroščevanje programa in pregledu trenutnega stanja registrov.

3.4 Temperaturni senzor DHT22

DHT22 (Slika 3.5) [11] je nizkocenovni senzor, ki omogoča merjenje temperature in vlage. Za merjenje uporablja kapacitivni senzor vlage. S senzorjem vlage lahko merimo od 0 - 100 % relativne vlažnosti, pri čemer je zagotovljena

DHT22 pins	
1	VCC
2	DATA
3	NC
4	GND

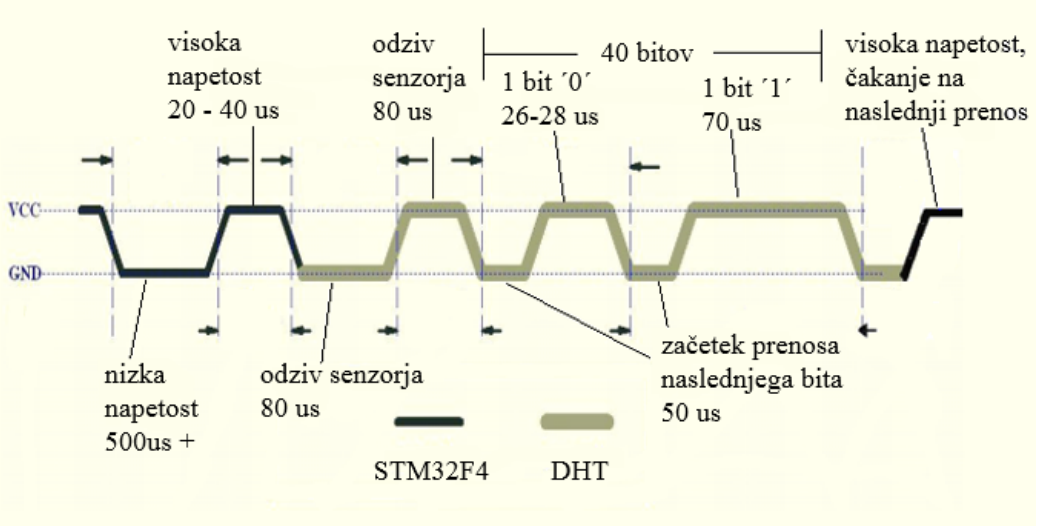


Slika 3.5: Senzor DHT22 in tabela vezave pinov.

natančnost med 2 - 5 %. Termistor DS18B20 pa se uporablja za merjenje zunanje temperature. Termistor ima merilni razpon od -40 °C do 125 °C z natančnostjo do pol stopinje Celzija. Senzor deluje s hitrostjo 0.5 Hz, kar pomeni, da lahko podatke iz njega pridobivamo vsaki dve sekundi. Za povezavo uporablja štiri pine (Slika 3.5). Prvega se uporablja za napajanje in ga lahko priklopimo na 3.3 - 6 V DC, drugega se uporablja za prenos podatkov, tretji ostane nepovezan, četrtega pa se priklopi na ozemljitev.

Serijska komunikacija s senzorjem

Začetek prenosa podatkov se začne s postavitvijo komunikacijskega pina v nizko napetostno stanje (Slika 3.6). V takem stanju čakamo 500 mikrosekund, nato postavimo pin v visoko napetostno stanje in počakamo še 20 do 40 mikrosekund. Za tem odmorom vodilo prevzame senzor in ponovno postavi vodilo v nizko napetostno stanje za 80 mikrosekund in ga nato vzdigne še za dodatnih 80 mikrosekund. Za tem se začne prenos podatkov. Pred pošiljanjem vsakega bita senzor za 50 mikrosekund postavi vodilo v nizko napetostno stanje, nato pa pošlje podatkovni bit. Vrednost bita se določi



Slika 3.6: Shema serijskega prenosa podatkov.

glede na čas, ko je vodilo ostalo v visokem napetostnem stanju. Za logično 0 mora senzor držati visoko napetost okoli 27 mikrosekund za logično 1 pa 70 mikrosekund. Po prenosu 40 bitov oziroma 5 bajtov STM prevzame vodilo in ga postavi v visoko napetostno stanje ter čaka na naslednji prenos podatkov. Iz pridobljenih petih bajtov podatkov prva dva uporabimo za izračun vlage, druga dva za izračun temperature, zadnji pa je redundanten in z njim preverimo pravilnost prenesenih podatkov.

3.5 Ethernet modul USR-TCP232-T

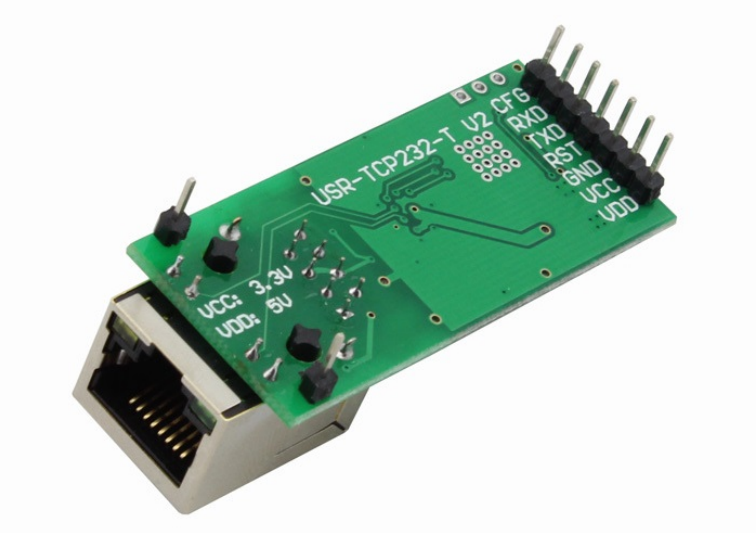
USR-TCP232-T (Slika 3.7) [12] je enostaven nizkocenovni ethernet modul z vgrajenim TCP/IP skladom. Mikrokrmilnik z modulom komunicira preko UART protokola in lahko uporablja boudrate med 300 in 25600 bps. Modulu lahko nastavimo štiri različne načine delovanja, in sicer lahko deluje kot TCP (*ang. Transmission Control Protocol*) strežnik, TCP odjemalec, UDP (*ang. User Datagram Protocol*) strežnik ali UDP odjemalec. Za povezavo modula potrebujemo vsaj pet žic. Dve žici potrebujemo za GND in 5V VDD, dve za

komunikacijo RX in TX ter eno za CFG pin, ki nam omogoča nastavljanje tega modula.

Ob postavitvi pina CFG v nizko stanje nam modul preko UART-a s parametri 9600 bps,n,8,1 pošlje znak 'U' in nam s tem pove, da je prešel v nastavitveni način.
Za tem je treba poslati naslednje podatke (IP naslove, porte in boudrate se pošilja v obratnem vrstnem redu !!!):

- predpona, 2B, 0x55 0xAA
- ciljni IP, 4B
- ciljni port, 2B
- modulov IP, 4B
- modulov port, 2B
- GW IP, 4B
- način delovanja, 1B, od 0x00 do 0x03
- baudrate, 3B
- nastavitev UART komunikacije, 1B,
  brez paritete, 8 bitov podatkov, 1 stop bit = 0x03
- redundanca, 1B, seštevek vseh bajtov brez predpone, upošteva se samo zadnji bajt

Ko modul prejme vse podatke, preko redundantnega bajta preveri, ali je prišlo do napake. Ob uspešni nastavitvi poslanih podatkov odgovori z znakom 'K', v primeru, da je prišlo do napake, pa vrne znak 'E'. Po odgovoru modula je treba CFG pin ponovno nastaviti v visoko napetostno stanje. Za tem lahko začnemo s komunikacijo. Ob prejetju podatkov preko UART-a modul glede na podane nastavitve sam opremi paket s potrebnimi glavami in jih pošlje v omrežje. V primeru, da modul dobi podatke iz omrežja, jih dekapsulira in jih preko UART-a posreduje mikrokrmilniku.



Slika 3.7: Ethernet modul USR-TCP232-T.

3.6 Ultrazvočni senzor MB7060

XL-MaxSonar®-WR1^{TM} MB7060 (Slika 3.8) [13] je ultrazvočni senzor, namenjen merjenju vodne gladine. Senzor je izdelan po IP67 standardu, kar pomeni, da je vodoodporen. Meri razdalje od 0 do 765 cm, pri čemer je zagotovljena enocentimetrska natančnost le od 20 cm naprej. Podatke iz senzorja lahko beremo s frekvenco 10 Hz, kar je enako desetim branjem na sekundo.

Notranjost vseh ultrazvočnih senzorjev je sestavljena iz oddajnika in prejemnika. Za opravljanje meritev mora najprej oddajnik oddati zvočni signal, ta potuje do najbližje ovire, kjer se odbije in vrne nazaj do senzorja, le-ta ga s pomočjo sprejemnika ulovi. Senzor nato na podlagi časa potovanja in hitrosti zvoka izračuna razdaljo od senzorja do merjenega objekta, ki bo v našem primeru voda.

MB7060 zaradi uporabe šumnih filtrov in avtomatske kalibracije dosega zelo veliko natančnost znotraj 9-centimetrskega merilnega snopa. Za branje podatkov imamo na razpolago prilagojeno RS232 komunikacijo preko analogne napetosti ali pa s širino pulza. Za delovanje potrebuje od 3 V do 5.5 V in nizek 3.4 mA povprečni tok. Za serijsko komunikacijo moramo povezati 6.

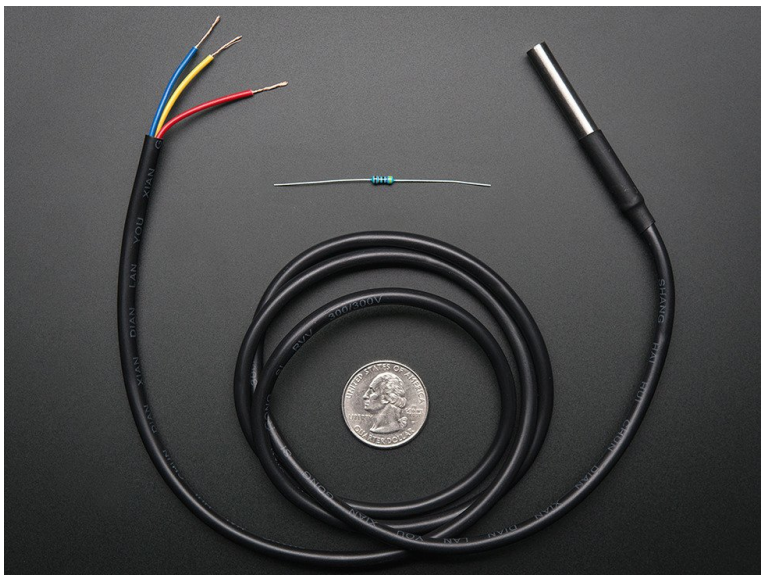
Slika 3.8: Ultrazvočni senzor MB7060.

pin na VDD, 7. pin na GND in 5. pin uporabimo za UART branje podatkov. Uporaba 4. pina je opcijska, saj z njim reguliramo, ali bo senzor konstantno opravljal meritve ali bomo sami določali čas njegovega delovanja. Senzor ima za UART komunikacijo prednastavljen baudrate na 9600 bps, pošilja 8 bitov podatkov, ne uporablja paritete in ima 1 stop bit. Preden začne pošiljati podatke, najprej pošlje znak 'R', nato pa tri številske znake, ki predstavljajo izmerjeno razdaljo v centimetrih.

3.7 Vodoodporni temperaturni senzor DS18B20

DS18B20 (Slika 3.9) [14] je digitalni temperaturni senzor, ki omogoča od 9 do 12-bitne temperaturne meritve. Poleg merjenja temperature nam ponuja še alarmiranje, ki ga je z zgornjo in spodnjo mejo mogoče nastaviti preko notranjih registrov. Senzor nam omogoča merjenje temperature od -55 °C do 125 °C, vendar je obljubljena natančnost +/-0.5 °C le znotraj razpona -10 °C do 85 °C. Za priklop senzorja potrebujemo dve žici, in sicer podatkovno in GND, vendar imamo v tem primeru omejeno zmogljivost delovanja. Če

želimo, da senzor deluje brez omejitev, mu je potrebno dodati še napajanje. Da je načrtovalcem senzorja uspelo priklopiti senzor na samo dve žici in so kljub temu omogočili priključitev večih senzorjev na en pin, so morali zasnovati svoj komunikacijski vmesnik 1-wire.



Slika 3.9: Vodoodporni temperaturni senzor DS18B20.

1-wire

1-wire vmesnik, kot nam že samo ime pove, poteka preko ene žice. Da je komunikacija nemotena, je potrebna natančna sinhronizacija in naslavljaje. Komunikacija v protokolu je razdeljena na tri dele. Prvi del je inicializacija in se začne z nizkim napetostnim pulzom, ki ga postavi mikrokrmilnik. Takoj za tem vsi senzorji, ki so priklopljeni na isto vodilo, odgovorijo z visokim napetostnim pulzom. Po uspešno izvedeni inicializaciji sledi drugi del, ki služi naslavljanju senzorja. Za naslavljaje posameznega senzorja se uporablja unikatna 64-bitna ROM koda. V drugem delu mikrokrmilnik najprej pošlje 2-bitni ROM ukaz, ki mu omogoča iskanje senzorjev, branje njihovih naslovov ali pa njihovo naslavljaje. Ko mikrokrmilnik vzpostavi povezavo s

točno določenim senzorjem, nastopi še zadnji del, ki služi izvajanju funkcijskih ukazov. S funkcijskimi ukazi mikrokrmilnik upravlja delovanje senzorja. Najpogosteje rabljen ukaz je branje scratchpada. S klicem tega ukaza se začne kopiranje 9-bajtnega notranjega pomnilnika. Pri čemer prva dva bajta predstavljata temperaturo, druga dva bajta sta alarmni vrednosti, peti je konfiguracijski register, naslednji trije so rezervirani za senzor, zadnji pa je CRC (*ang. Cycle redundancy check*) in se ga uporablja za preverjanje pravilnosti prenosa. Drugi ukaz, ki je še pogosto uporabljen, je ukaz za začetek pridobitve temperature. To traja od 100 ms za 9-bitno resolucijo in do 750 ms za 12-bitno resolucijo. Za nastavitev alarmnih vrednosti in za spreminjanje resolucije meritev se uporablja ukaz za pisanje na scratchpad. Ostali ukazi so še branje napajanja in kopiranje na scratchpad.

3.8 Senzor za merjenje raztopljenega kisika v vodi

Atlas EZO Dissolved Oxygen kit (Slika 3.10) [15] je senzor, namenjen merjenju vsebnosti raztopljenega kisika. Za delovanje senzorja so potrebni trije deli, in sicer merilna sonda, ki se potopi v vodo, vtič za priklop sonde na modul in modul za komunikacijo z mikrokrmilnikom. Senzor omogoča meritve od 0.01 do 36 mg/l z odstopanjem +/- 0.2 mg/l. Poleg izvajanja samih meritev senzor ponuja tri različne kalibracije. Kljub temu da omogoča kalibriranje temperature, pritiska in slanosti, bomo v diplomski nalogi kalibrirali samo temperaturo, ker ima največji vpliv na vsebnost kisika in je edini parameter izmed navedenih treh, ki se bo skozi čas spreminjal. Ostali dve kalibraciji ne bosta potrebni, saj ima senzor prednastavljeno slanost na sladko vodo, globina potopljene sonde pa ne bo večja od 10 metrov in s tem ne bo vplivala na meritve. Senzor za komunikacijo z mikrokontrolerjem ponuja protokola I2C in UART in uporablja od 3 do 5 V napetost. Za potrebe diplomske naloge smo se odločili za uporabo UART protokola in s tem poenotili način komunikacije z razvojno ploščico STM. Modul za UART komunikacijo uporablja

9600 bps baudrate, 1 stop bit in nobenega paritetnega bita.

Slika 3.10: Senzor za merjenje vrednosti raztopljenega kisika v vodi Atlas Ezo Dissolved Oxygen Kit.

Senzor, kot smo že prej omenili, lahko meri vsebnost kisika in tri različne kalibracije. Omogoča pa tudi dodatne funkcionalnosti, kot je računanje nasičenosti kisika, nastavitev nekonstantnega opravljanja meritev in pridobivanje informacij o delovanju modula. Od dodatnih funkcionalnosti smo rabili samo nastavitev meritev, saj se nam je zdelo nepotrebno opravljati konstantne meritve, poleg tega pa bi s tem bistveno povečali porabo energije. Za izklop konstantnih meritev je potrebno senzorju preko UART-a poslati ukaz ‘‘C,0<CR>’’, nakar nam senzor odgovori z ‘‘*OK<CR>’’. Po izklopu kontinuiranih meritev je za pridobitev meritve najprej potrebno poslati ukaz ‘‘R<CR>’’.

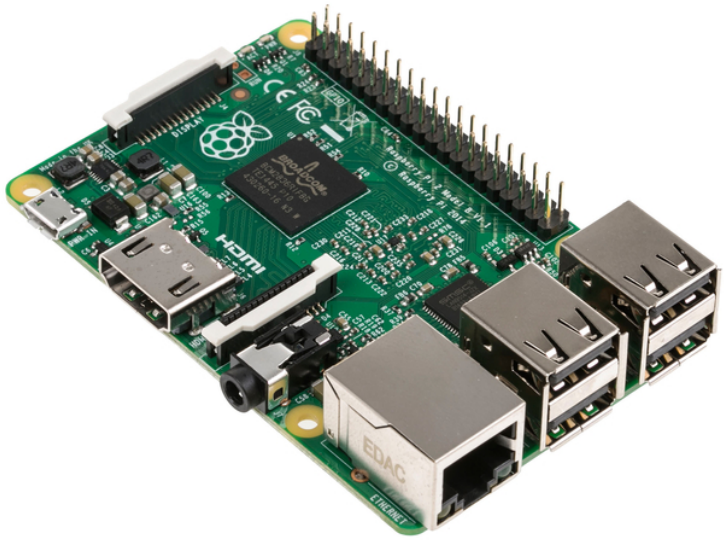
Poglavje 4

Opis tehnologij in orodij za izdelavo strežnika

Poleg vgrajenega sistema, ki skrbi za odvzem podatkov, smo morali implementirati tudi strežnik, ki bo skrbel za hranjenje pridobljenih podatkov in prikaz teh podatkov uporabniku. Za prikaz podatkov smo se odločili narediti enostavno spletno stran. Za hranjenje in prikaz podatkov smo uporabili Raspberry Pi, ki je cenovno ugoden in ponuja vsa potrebna orodja. Za delovanje Raspberry-a v načinu strežnika smo uporabili strežniški program Apache, MySqlServer in PHP 5+.

4.1 Raspberry Pi 2 Model B

Raspberry pi je nizkocenovni majhen računalnik, ki je bil razvit v izobraževalne namene. Zaradi slednjega in relativno nizke cene je dostopen širši množici. Zaradi nizke cene in dokaj visoke zmogljivosti ima zelo široko uporabnost in ga lahko najdemo v najrazličnejših projektih. V diplomski nalogi ga bomo uporabili kakor stražnik za odvzem, hranjenje in prikaz podatkov. Odločili smo se za uporabo zadnje različice Raspberry-a, in sicer Raspberry Pi 2 Model B (Slika 4.1) [16], ki se od svojega predhodnika razlikuje v procesorski moči, saj uporablja 900MHz štirijedrni procesor ARM Cortex A7, poleg tega



Slika 4.1: Raspberry Pi 2 Model B.

pa vsebuje 1 GB rama. Zaradi uporabe novega procesorja lahko sedaj na Raspberry Pi naložimo tudi kompleksnejše operacijske sisteme, kot je Windows 10. Mi smo se zaradi boljše zmogljivosti odločili za enostavnejši in bolj podprt operacijski sistem Raspbian, ki temelji na Linux distribuciji Debiana. Za povezovanje imamo na voljo 4 USB porte, 1 hdmi, 1 audio izhod, ethernet port, 1 vhod za SD kartico ter 40 splošno namenskih pinov.

4.2 GSM\GPRS modul Adafruit FONA

Adafruit FONA (Slika 4.2) [17] je GSM/GPRS modul, ki nam omogoča glasovno klicanje, pošiljanje SMS-ov, povezovanje z GPRS omrežjem in poslušanje FM radia. Za delovanje modula skrbi SIMCom-ov čip SIM800, ki je v velikosti poštne znamke.

Čip SIM800 omogoča:

- štiripasovno povezavo 850/900/1800/1900 MHZ, ki omogoča povezavo na vsa GSM omrežja z 2G podporo,

- sprejemanje in kreiranje govornih klicev z uporabo slušalk ali zunanjega 8 Ω zvočnika in mikrofona,
- prejemanje in pošiljanje SMS sporočil (*ang. short message service*),
- prejemanje in pošiljanje podatkov preko GPRS povezave (TCP/IP, HTTP),
- pregled in poslušanje FM radio kanalov,
- PWM za priklop brenčala (*ang. buzzer*).

Slika 4.2: GSM modul Adafruit FONA.

Za upravljanje modula se uporabljajo standardni AT ukazi, ki se jih modulu pošilja preko UART protokola. Da je uporaba še lažja, ima modul samodejno zaznavanje baudrate-a, kar pomeni da mu lahko pošiljamo podatke s poljubno hitrostjo, ki jo modul sam zazna brez predhodnih nastavitev.

Modul poleg običajnih pin izhodov in vhodov ponuja še 3.5 mm avdio vtič, ki se ga uporablja za priklop slušalk, zvočnika ali mikrona. JST 2-pin

za priklop baterije, ki je obvezna ter nam omogoča delovanje brez stalnega napajanja. Za polnjenje baterije se uporablja mikro USB, zraven so vključili še dve led diodi, ki služita prikazu stanja polnjenja. Za povezavo modula z omrežjem je obvezno potrebna še antena, ki se jo priključi preko SMA vhoda. Adafruit fona za povezavo z mikrokontrolerjem potrebuje vsaj šest žic, dve za povezavo ozemljitve (GND) in 3 do 5 V napajanja (VIO), dve za UART komunikacijo (TX, RX), KEY za upravljanje delovanja in RST za reset modula.

4.3 Apache

Apache je odprto kodni spletni strežnik, ki je bil na začetku baziran na NSCA HTTPd strežniku. Prva različica je bila izdana leta 1995. Apache je igral ključno vlogo v razvoju svetovnega spleta in je bil prvi strežnik, ki je presegel mejo 100 milijonov streženih spletnih strani. Trenutno ima največji delež uporabe med strežniki oz. se nekaj več kot 50 % spletnih strani prikazuje preko Apache strežnika. Apache je najpogosteje nameščen na Linuxovih strežnikih, ga je pa mogoče namestiti na vse večje operacijske sisteme. Poleg prikaza običajnih spletnih strani omogoča še prenos datotek, PHP (*ang. Hypertext Preprocessor*) strežnik, SSL (*ang. secure sockets layer* povezavo) itd.

4.4 RRDTool

RRDtool [21] je odprto kodni industrijski standard za beleženje in izrisovanje časovno zaporednih podatkov. RRDtool temelji na RRD (*ang. round robin databse*) bazi s krožnim dodeljevanjem, kar omogoča predstavljeno velikost baze, ki se s časom ne povečuje. Orodje je mogoče uporabljati kot samostojno znotraj linux terminala, omogoča pa integracijo v Perl, Python, Ruby, PHP in Lua.

Beleženje podatkov

RRDtool beleži podatke glede na časovno značko. Ob kreiranju baze moramo navesti časovni dotok podatkov in kakšno odstopanje je še sprejemljivo. V primeru, da v danem časovnem okviru RRDtool ne dobi podatka, si shrani neznano vrednost namesto številke, kajti v nasprotnem primeru bi lahko prišlo do napačne interpretacije v grafih. Za hranjenje podatkov RRDtool uporablja RRA (*ang. round robin archive*) arhiv s krožnim dodeljevanjem. Vsaka RRDtool baza mora vsebovati vsaj en RRA. Za uporabo RRA je treba nastaviti, kakšno vrednost si bo pomnil. Kljub možnosti uporabe še minimalne kot maksimalne vrednosti smo za potrebo diplomske naloge uporabili samo povprečno vrednost. Za določanje vrednosti je potrebno navesti še obdobje, znotraj katerega se izračuna vrednost, ter koliko takšnih obdobij bomo pomnili, preden jih baza začne prepisovati.

4.5 MySql

MySql je relacijska podatkovna baza, katere prva različica je bila izdana leta 1995. Trenutno za razvoj in distribucijo MySql-a skrbi podjetje Oracle. MySql je zelo priljubljen med razvijalci spletnih aplikacij in je ena izmed ključnih komponent WAMP-a (*ang. Windows Apche Mysql Perl/PHP/Python*) in LAMP-a (*ang. Linux Apche Mysql Perl/PHP/Python*), poleg tega pa je MySql podprt za vse večje operacijske sisteme. MySql je napisan v jeziku C in C++ in je bil narejen za uporabo brez grafičnega vmesnika. Zaradi lažje uporabe se lahko namesti phpMyAdmin ali Oraclov MySql Workbanch, ki ponujata grafični vmesnik z dodatnimi orodji za razvoj podatkovne baze. Za kreiranje, upravljanje, polnjenje in prikazovanje baze se uporablja poizvedovalni jezik SQL (*Structured Query Language*).

4.6 MySql Workbench

MySql Workbench je razvojni in administrativni grafični vmesnik za delo z MySql bazami. Je odprto koden program, napisan v programskem jeziku C in C++, ki ga razvija računalniško podjetje Oracle. Orodje nam preko avtomatiziranih postopkov pomaga načrtovati in kreirati MySql bazo, ki jo je nato enostavno sinhronizirati na strežnik. Omogoča nam tudi grafično sestavljanje SQL ukazov, kar nam zelo olajša delo. Poleg tega nam nudi celoten pregled nad stanjem strežnika ter njegovo upravljanje.

4.7 PHP

PHP je strežniški skriptni jezik za razvoj spletnih strani, ki se ga lahko uporablja kot samostojni programski jezik. Nastal je leta 1995 kot pomoč za urejanje spletne strani. Leta 2013 pa je bil uporabljen na več kot 240 milijonih spletih strani in 2 milijonih spletnih strežnikov. Njegova prednost je enostavno mešanje PHP kode s HTML (ang. hyper text markup language) kodo in enostavna sintaksa.

4.8 HTML

HTML je standarden jezik za izdelavo spletnih strani. Nastal je leta 1993, medtem ko je zadnja uradna različica HTML 5.0 izšla leta 2014. HTML ni programski jezik, ampak označevalni jezik, saj opisuje strukturo spletne strani ter način njenega prikaza. HTML datoteke so sestavljene iz značk, ki definirajo strukturo spletne. V posamezne bloke, ki jih definiramo z značkami, se lahko vključi različne strukture, kot so slike in objekti, omogoča pa še dodajanje Javscripta, PHP-ja in drugih skript.

4.9 CSS

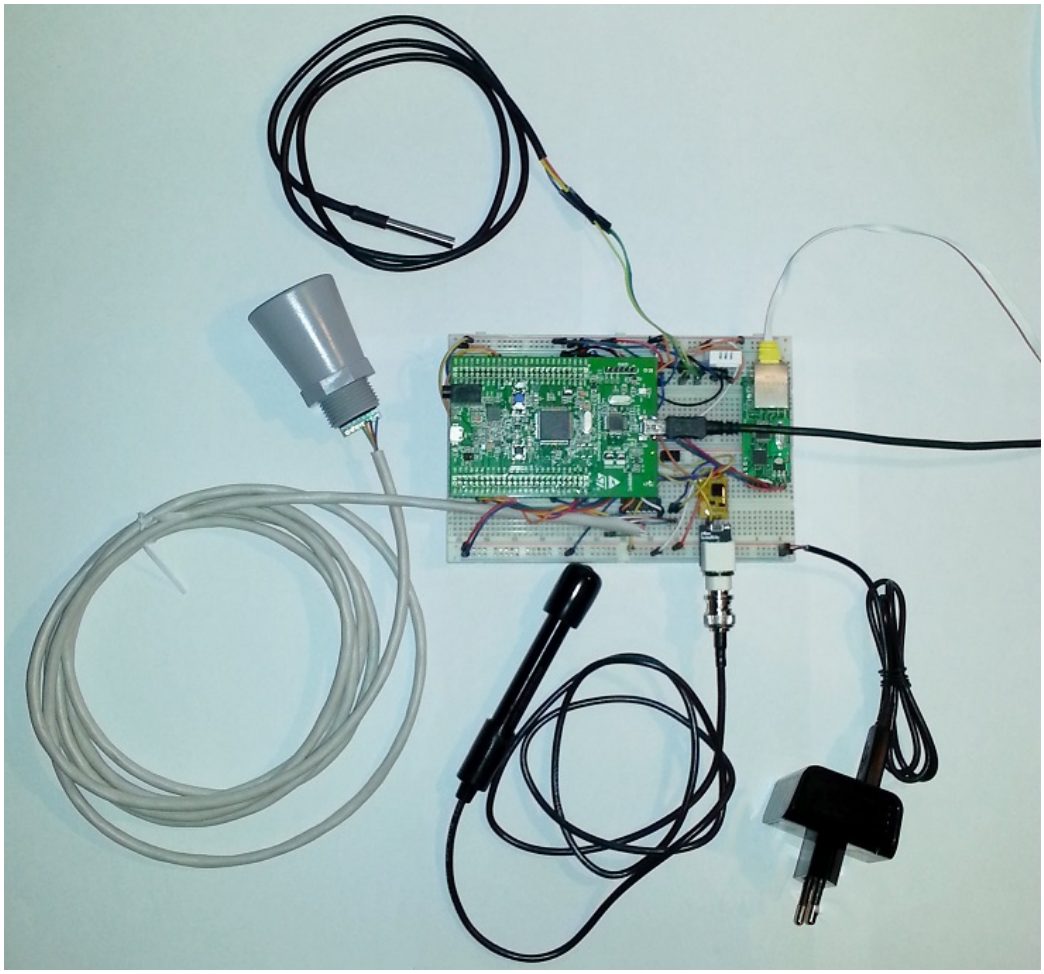
CSS (*ang. cascading style sheet*) je jezik, ki se uporablja za opisovanje formata in izgleda spletnih strani. Poleg HTML-ja in Javascripta je CSS eden izmed temeljnih jezikov za izdelavo spletnih strani, aplikacij in uporabniških vmesnikov za mobilne aplikacije. Uporaba CSS je enostavna, saj nam omogoča uporabo ene datoteke za oblikovanje celotne spletne strani, sestavljene iz različnih HTML. Posledično nam olajša tudi spreminjanje izgleda spletne strani, saj je potrebno to spremeniti le na enem mestu.

Poglavje 5

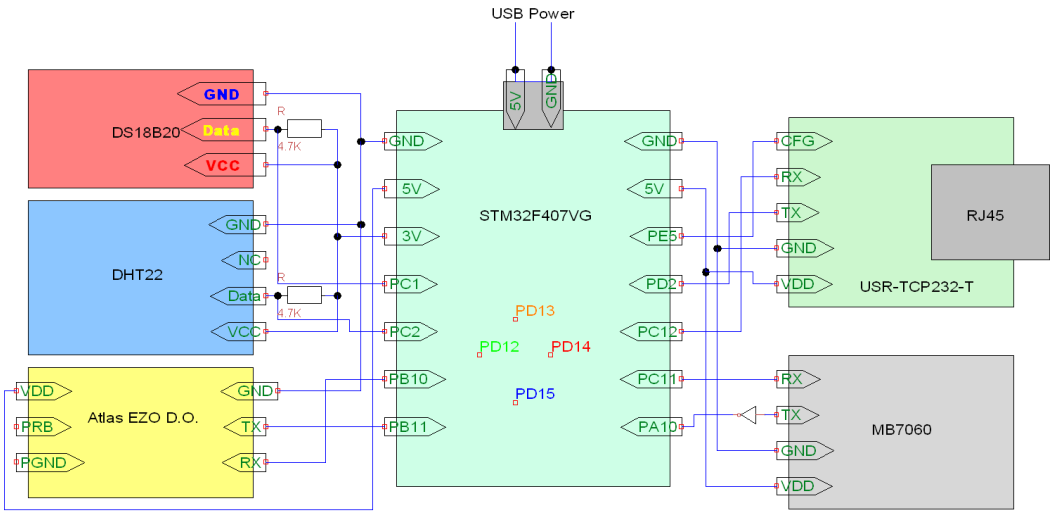
Načrtovanje in izdelava vgrajenega sistema

V prvem delu diplomske naloge smo opisali vsa orodja in senzorje, ki smo jih uporabili za izdelavo našega sistema. V drugem delu pa bomo bolj natančno navedli, kako smo se lotili razvoja ter kako ta deluje. Programsko kodo za razvojno ploščico STM32F4 Discovery smo pisali v namenskem programskem okolju IAR Embedded Workbenck. Za razvoj smo uporabili brezplačno verzijo programa, ki je omejena s končno velikostjo projekta. Vsa razvita koda za STM32F4 je bila napisana v programskem jeziku C. Poleg kode, ki smo jo napisali sami, smo uporabili še splošne knjižnice in strojno-programsko opremo, ponujeno s strani izdelovalca ploščice.

Na STM smo preko žic priklopili prej opisane senzorje in iz njih brali podatke. Pridobljene podatke smo preko ethernet modula poslali na strežnik, ki ga bomo v nadaljevanju podrobneje opisali. Večina uporabljenih modulov in senzorjev je s STM-om komunicirala preko U(S)ART-a, za ostale pa je bilo treba spisati prilagojeno serijsko komunikacijo, ki jo bomo v nadaljevanju natančno opisali. Na spodnji sliki (Slika 5.2) je prikazana natančna shema vezave modulov in senzorjev z razvojno ploščico STM.



Slika 5.1: Slika vgrajenega sistema z vsemi senzorji.



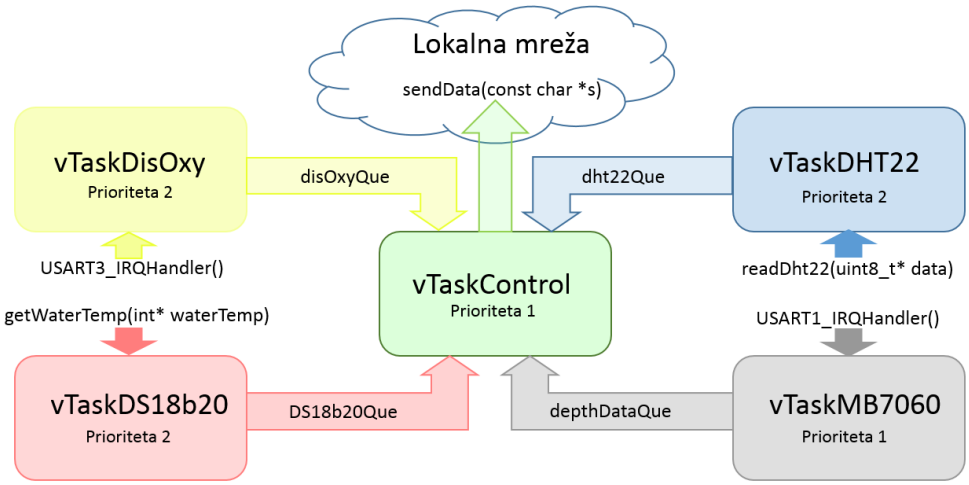
Slika 5.2: Vezava vgrajenega sistema.

5.1 Kreiranje opravil in izmenjava podatkov

Za lažje in varčnejše obvladovanje sistema smo se odločili za uporabo opravil. Opravila smo ustvarili s pomočjo operacijskega sistema FreeTROS. Zaradi različnih prioritet smo opravila ločili na dve prioritetni skupini. V prvi skupini imamo dve opravili, ki sta bili po našem mnenju najbolj rizični, zato smo jima dodelili najvišjo prioriteto. Prvo opravilo z najvišjo prioriteto skrbi za pošiljanje podatkov na server, drugo pa je zadolženo za zajem podatkov globine bazena. Opravila z nižjo prioriteto pa skrbijo za merjenje temperature zraka in vode, vsebnosti raztopljenega kisika v vodi in vlage.

Delovanje opravil

Opravila, ki so zadolžena za pridobivanje podatkov iz senzorjev, najprej izmerijo vrednost, ki jo nato preko vrste posredujejo glavnemu opravilu. Vsako opravilo po meritvi prepiše svoje podatke v svojo vrsto, ki si jo deli z glavnim opravilom. Podatke je obvezno treba prepisati, saj bi ob uporabi referenc



Slika 5.3: Shema delovanja opravil.

lahko prišlo do prenosa napačnih podatkov. Glavno opravilo po določeni urini periodi prebere podatke, ki se nahajajo v vrstah in jih preko ethernet modula pošlje na strežnik.

5.2 Nastavitev U(S)ART protokola

Zaradi večkratne uporabe U(S)ART-a smo se odločili, da ga bomo posebej opisali. Kot smo že v teoretičnem delu naloge (3) omenili, je za nastavitev U(S)ART-a treba poznati štiri parametre, in sicer s kakšno hitrostjo se pošilja/prejema, koliko podatkov pošljemo zaporedoma, ali uporabimo paritetni bit in koliko stop bitov uporabimo. V našem primeru so vsi moduli, ki so komunicirali preko U(S)ART-a, uporabljali iste nastavitve, zato bomo opisali samo enega. Poleg same nastavitve U(S)ART-a je na STM potrebno nastaviti še pine, preko katerih poteka komunikacija in prekinitev, ki nam omogoča lažje branje podatkov.

Nastavitev U(S)ART parametrom

Kot smo že prej omenili, so v našem primeru vsi moduli uporabljali iste parametre, zato bomo opisali samo nastavitev U(S)ART3 kanala, ki smo ga uporabili za komunikacijo s senzorjem za merjenje kisika. Za komunikacijo je bilo potrebno baudrate nastaviti na 9600 bps, pošilja se 8 bitov zaporednih podatkov in en stop bit. Pri komunikaciji se ne uporablja paritetnega bita.

```
1
2  //funkcija za inicializacijo U(S)ART3 kanala
3  void init_USART3(){
4    //dodelitev ure
5      RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);
6    //nastavitev parametrov
7      USART_InitTypeDef USART_InitStruct;
8      USART_InitStruct.USART_BaudRate = 9600;
9      USART_InitStruct.USART_WordLength = USART_WordLength_8b;
10     USART_InitStruct.USART_StopBits = USART_StopBits_1;
11     USART_InitStruct.USART_Parity = USART_Parity_No;
12     USART_InitStruct.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
13     USART_InitStruct.USART_HardwareFlowControl=
14     USART_HardwareFlowControl_None;
15
16   //inicializacija
17     USART_Init(USART3, &USART_InitStruct);
18   //vklop U(S)ART3 kanala
19     USART_Cmd(USART3, ENABLE);
20 }
```

Nastavitev pinov

Da omogočimo U(S)ART-u komunikacjio z moduli, mu je treba dodeliti pine. STM ima točno določeno, kateri pini pripadajo kateremu kanalu, zato spodaj

prilagamo tabelo (Tabela 5.2), v kateri je prikazano, katere pine smo uporabili. Za uporabo pinov v U(S)ART komunikaciji je treba nastaviti način na GPIO_Mode_AF, kar pomeni, da se uporabi alternativno funkcijo za upravljanje pinov.

U(S)ART kanal	***TX***	***RX***
U(S)ART1	**PA9**	**PA10**
U(S)ART3	**PB10**	**PB11**
UART5	**PC12**	**PD2**

Tabela 5.1: U(S)ART komunikacijski pini.

```
1
2  //funkcija za inicializacijo U(S)ART3 pinov
3  void init3_GPIO_AF() {
4    //dodelitev ure
5     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
6    //nastavitev parametrov
7     GPIO_InitTypeDef GPIO_InitStruct;
8     GPIO_InitStruct.GPIO_Pin = GPIO_Pin_11 | GPIO_Pin_10;
9     GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
10    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
11    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
12    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
13   //inicializacija
14    GPIO_Init(GPIOB, &GPIO_InitStruct);
15   //vklop U(S)ART3 pinov
16    GPIO_PinAFConfig(GPIOB, GPIO_PinSource10, GPIO_AF_USART3);
17    GPIO_PinAFConfig(GPIOB, GPIO_PinSource11, GPIO_AF_USART3);
18 }
```

Uporaba prekinitev za branje

Za lažje pridobivanje podatkov prek U(S)ART-a smo se odločili za uporabo prekinitev, saj bi morali v nasprotnem primeru stalno izvajati kodo za zajem podatkov. Za uporabo prekinitev je naprej potrebno prekinitev inicializirati znotraj glavnega progama, nato pa moramo še dodati izvajalno kodo v datoteko s prekinitvami. Ker se prekinitev nahaja v drugi datoteki, je treba pri inicializaciji spremenljivke dodati predpono extern, kar nam omogoči uporabo spremenljivk iz drugih C datotek.

```
1
2  //spremenljivke za izmenjavo podatkov med datotekami
3  uint16_t txtDO[10];
4  extern int countData3;
5
6  //funkcija za inicializacijo U(S)ART3 prekinitve
7  void pspUsart3(){
8    //nastavitev parametrov
9     NVIC_InitTypeDef NVIC_InitSTructure;
10    NVIC_InitSTructure.NVIC_IRQChannel=USART3_IRQn;
11    NVIC_InitSTructure.NVIC_IRQChannelPreemptionPriority=0;
12    NVIC_InitSTructure.NVIC_IRQChannelSubPriority=1;
13    NVIC_InitSTructure.NVIC_IRQChannelCmd=ENABLE;
14   //inicializacija
15    NVIC_Init(&NVIC_InitSTructure);
16   //vklop U(S)ART3 prekinitve ki skrbi za sprejem podatkov
17    USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);
18 }
```

Spodnji izsek programske kode se mora nahajati znotraj stm32f4xx_it.c datoteke, v nasprotnem primeru ne pride do izvajanja prekinitve.

```
1
2  //spremenljivke za izmenjavo podatkov med datotekami
3  volatile uint16_t dataUsart3;
4  volatile int countData3=0;
5  extern volatile uint16_t txtDO[10];
6
7  //prekinitvena funkcija, ki se izvede ob prihodu podatkov prek RX
       pina
8  void USART3_IRQHandler(void){
9    //preveri ali so je prenos bajta zakljucil
10   if(USART_GetITStatus(USART3,USART_IT_RXNE)){
11    //reset ineksa da ne pride SEGMENTATION FAULT-a
12       if(countData3==10){
13         countData3=0;
14       }
15     //sprejem podatka
16       dataUsart3=USART_ReceiveData(USART3);
17     //zapis podatka v vektor
18       txtDO[countData3]=dataUsart3;
19     //povecanje indeksa
20       countData3++;
21     //preveri ce je konec podatkovnega Stringa
22       if(dataUsart3==0x0D){
23         countData3=0;
24       }
25     //vracanje iz prekinitve
26       USART_ClearITPendingBit(USART3,USART_IT_RXNE);
27   }
28 }
```

5.3 Nastavitev in uporaba DHT22

Za uporabo senzorja DHT22 smo naredili svoje opravilo, ki enkrat na uro prebere vrednost. Poleg tega je bilo treba napisati še funkcijo za serijsko branje podatkov, saj senzor ne uporablja splošnih komunikacijskih protokolov. Za povezavo senzorja smo rabili štiri žice, in sicer eno za GND, eno za 3.3 V, dodatni dve pa za branje podatkov, saj je potrebno uporabiti 4.7 K Ω upor. Vezava samega senzorja je prikazana na shemi (Slika 5.2).

vTaskDHT22

vTaskDHT22 skrbi za zajem podatkov, izračun vrednosti in posredovanje glavnemu opravilu preko dht22Que vrste. Ker to opravilo nima ključnega pomena pri nadzoru ribogojnice, ima prioriteto 2.

```
1
2  void vTaskDHT22 (void *pvParameters){
3    //inicializacija spremenljivk
4       int readDataDht[2]={0,0};
5
6       while(1){
7           //eno urni premor
8            vTaskDelay(3598500 / portTICK_RATE_MS);
9          //branje podatkov
10           a=readDht22(data);
11         //preverjanje pravilnosti in ponovno branje v primeru
               napake
12           while(a!=1){a=readDht22(data);}
13
14        //izracun temperature zraka in vlage iz pridobljenih
              podatkov
15        //HUM case DHT22:
16           readDataDht[0] = data[0] & 0x7F;
```

```
17          readDataDht[0] *= 256;
18          readDataDht[0] += data[1];
19        //TEMP case DHT22:
20          readDataDht[1] = data[2];
21          readDataDht[1] *= 256;
22          readDataDht[1] += data[3];
23      //posiljanje v vrsto
24          xQueueSendToBack(dht22Que,readDataDht,portMAX_DELAY);
25      }
26  }
```

Branje podatkov

Za pošiljanje podatkov senzor uporablja svoj serijski protokol, ki smo ga opisali v teoretičnem delu (3). Vso komunikacijo s senzorjem smo opravljali preko pina PC2. Ker se je uporabljal samo en pin, je bilo treba med komunikacijo izmenjevati način delovanja pina, zato smo napisali še dve dodatni funkciji. Poleg tega je bilo treba napisati še mikrosekundo pavzo, ki pa jo v diplomsko nalogo nismo vključili.

```
1
2   //nastavitev pina kot vhod
3   void SetTempPinAsInput(){
4      pc2.GPIO_Pin = GPIO_Pin_2;
5           pc2.GPIO_Mode = GPIO_Mode_IN;
6           pc2.GPIO_OType = GPIO_OType_PP;
7           pc2.GPIO_Speed = GPIO_Speed_100MHz;
8           pc2.GPIO_PuPd = GPIO_PuPd_NOPULL;
9           GPIO_Init(GPIOC, &pc2);
10  }
11  //nastavitev pina kot izhod
12  void SetTempPinAsOutput(){
```

```
13    pc2.GPIO_Pin = GPIO_Pin_2;
14         pc2.GPIO_Mode = GPIO_Mode_OUT;
15         pc2.GPIO_OType = GPIO_OType_PP;
16         pc2.GPIO_Speed = GPIO_Speed_100MHz;
17         pc2.GPIO_PuPd = GPIO_PuPd_NOPULL;
18         GPIO_Init(GPIOC, &pc2);
19 }
20 //branje podatkov iz senzorja
21 uint8_t readDht22(uint8_t* data){
22   //vklop casovnika za delay
23     TIM_Cmd(TIM3,ENABLE);
24   //inicializaciji spremenljivk
25     int i;
26     int volatile cnt=0;
27    data[0] = data[1] = data[2] = data[3] = data[4] = 0;
28   //postavitev linije v nizko napetostno stanje za ~20 ms
29     SetTempPinAsOutput();
30    GPIO_ResetBits(GPIOC, GPIO_Pin_2);
31    delayMs(18);
32   //postavitev linije v visoko napetostno stanje za 20us - 40us
33    GPIO_SetBits(GPIOC, GPIO_Pin_2);
34    delayUs(35);
35   //cakanje na odgovor senzorja
36     SetTempPinAsInput();
37     while(GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_2) == 1);
38    while(GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_2) == 0){
39      cnt++;
40      delayUs(1);
41      if(cnt>85){
42       return -1;
43      }
44    }
45    cnt=0;
```

```
46   while(GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_2) == 1){
47     cnt++;
48     delayUs(1);
49     if(cnt>85){
50      return -1;
51     }
52   }
53  //branje 40 bitov podatkov
54   for ( i=0; i< 40; i++) {
55   //50us nizko napetostno stanje pred bitom
56     while(GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_2) == 0){
57      cnt++;
58      delayUs(1);
59      if(cnt>60){
60        return -1;
61      }
62     }
63   //visoko napetostno stanje za 30us (12us) za bit='0' in 70us
        (35us) za bit="1"
64     cnt=0;
65     while(GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_2) == 1){
66      cnt++;
67      delayUs(1);
68      if(cnt>80){
69        return -1;
70      }
71     }
72   //dodaj '1' ali '0' in zamakni v desno
73     if(cnt<20){
74      data[i/8]=data[i/8]<<1;
75     }
76     else{
77      data[i/8]=data[i/8]<<1;
```

```
78      data[i/8]=data[i/8]|1;
79     }
80    }
81   //visoko napetostno stanje za izklop senzorja
82    SetTempPinAsOutput();
83   GPIO_SetBits(GPIOC, GPIO_Pin_2);
84   //izklop casovnika
85    TIM_Cmd(TIM3,DISABLE);
86   //preveri ce je prislo do napake pri prenosu
87    return (((data[0] + data[1] + data[2] + data[3]) == data[4]));
88 }
```

5.4 Nastavitev in uporaba DS18B20

Za meritev temperature vode smo uporabili senzor DS18B20. Meritve se opravljajo znotraj opravila vTaskDS18b20, podatke pa se posreduje glavnemu opravilu preko vrste DS18b20Que. Senzor primarno opravlja meritve z 12-bitno resolucijo, kar se nam je zdelo v redu in nismo spreminjali. Da opravi 12-bitno meritev, potrebuje 750 ms, kar za nas ne predstavlja problema, saj izvajamo meritev temperature enkrat na uro. Za komunikacijo senzor uporablja protokol 1-wire, ki je bil opisan v teoretičnem delu (3). Vsa komunikacija se vrši preko pina PC1. Vezava senzorja DS18B20 je enaka kot pri DHT22 in je prikazana na shemi (Slika 5.2). Za odvzem podatkov iz senzorja smo napisali svojo funkcijo getWaterTemp, ki implementira 1-wire protokol. Te kode zaradi obširnosti nismo vključevali v diplomsko nalogo. Po odvzemu podatkov s klicem funkcije getWaterTemp si podatke shranimo še v začasno spremenljivko waterTempStat, ki se uporablja za kalibracijo senzorja za raztopljeni kisik.

```
1
2  //opravilo za merjenje temperature vode
3  void vTaskDS18b20 (void *pvParameters){
4    //inicializacija spremenljivk
5     int waterTemp[2]={10,12};
6     int indT;
7     while(1){
8       //1 urna pavza
9          vTaskDelay(3598500 / portTICK_RATE_MS);
10         indT=0;
11       //funkcija za pridobivanje podatkov
12         startConversionDs18b20();
13       //branje podatkov
14         a2=getWaterTemp(waterTemp);
15      //ponovno branje v primeru napake
16         while(a2!=1 && indT<10){
17            a2=getWaterTemp(waterTemp);
18            vTaskDelay(50 / portTICK_RATE_MS);
19            indT++;
20         }
21       //shrani podatek za kalibracijo kisikovega senzorja
22         waterTempStat=waterTemp[0];
23       //posiljanje podatkov v vrsto
24         xQueueSendToBack(DS18b20Que,&waterTemp,portMAX_DELAY);
25     }
26 }
```

5.5 Nastavitev in uporaba Atlas EZO Dissolved Oxygen kita

Opravilo vTaskDisOxy, ki skrbi za delovanje Atlas EZO D.O. senzorja, je malo daljše od ostalih, saj vsebuje še kodo za kalibracijo senzorja, v kolikor pride do spremembe temperature vode. Za normalno delovanje opravila je bila potrebna predhodna inicializacija USART3 kanala. Na začetku opravila, preden vstopimo v glavno zanko, je treba senzorju najprej poslati ukaz za izklop kontinuiranega opravljanja meritev, saj ga ne potrebujemo, ker opravljamo meritve enkrat na minuto, poleg tega pa bi se nam povečala poraba energije. Po vstopu v glavno zanko se enkrat na minuto izvede meritev vrednosti raztopljenega kisika, ki se nato preko vrste disOxyQue posreduje glavnemu opravilu. Pred vsako meritvijo se opravi še pregled spremembe temperature in se v primeru odstopanja izvede kalibracijo. Temperaturna kalibracija senzorja je obvezna, saj je vsebnost kisika odvisna od trenutne temperature.

```
1
2  //opravilo za merjenje vsebnosti raztopljenega kisika
3  void vTaskDisOxy (void *pvParameters){
4    //inicializacija spremenljivk
5      char oxyValue[4];
6      char controlTxt3[3]={'C',',','O'};
7      char controlTxt2[3]={'C',',','?'};
8      char getDOdata[1]={'R'};
9      int curWaTemp=10;
10     int x,num,i;
11   //izklop konstantnega merjenja podatkov
12     while(1){
13       sendToDOSensor(controlTxt3);
14       vTaskDelay(2000 / portTICK_RATE_MS);
15       //if(txtDO[countData3-3]=='O' && txtDO[countData3-2]=='K'){
```

```
16      if(txtDO[6]=='O' && txtDO[7]=='K'){
17       break;
18     }
19   }
20  //preverjanje nastavitev
21   sendToDOSensor(controlTxt2);
22   vTaskDelay(2000 / portTICK_RATE_MS);
23  // glavna zanka
24   while(1){
25     //1 minutna pavza
26       vTaskDelay(58900 / portTICK_RATE_MS);
27     //kalibracija senzorja ob spremembi temperature vode
28       if(curWaTemp!=(int)waterTempStat && waterTempStat>0){
29         curWaTemp=waterTempStat;
30         num=waterTempStat;
31       //priprava podatkov za kalibracijo
32         controlTxt[0]='T';
33         controlTxt[1]=',';
34         if(num<100){x=2;}
35         if(num<10){x=1;}
36         for(i=0;i<x;i++){
37             controlTxt[1+x-i]=num%10+48;
38             num=num/10;
39         }
40         if(x==1){
41             controlTxt[3]=0;
42         }
43       //kalibriranje senzorja
44         while(1){
45           sendToDOSensor(controlTxt);
46           vTaskDelay(1000 / portTICK_RATE_MS);
47           if(txtDO[6]=='O' && txtDO[7]=='K'){
48             break;
```

```
49            }
50          }
51        }
52      //ukaz za zajem meritve
53        sendToDOSensor(getDOdata);
54        vTaskDelay(2000 / portTICK_RATE_MS);
55      //priprava podatkov za posiljanje
56        for(i=0;i<4;i++){
57          oxyValue[i]=txtDO[i];
58        }
59      //posiljanje podatkov v vrsto
60        xQueueSendToBack(disOxyQue,oxyValue,portMAX_DELAY);
61    }
62  }
```

5.6 Nastavitev in uporaba MB7060

Za merjenje globine smo uporabili opravilo vTaskMB7060. Temu opravilu smo dodelili najvišjo prioriteto, saj ima globina vode v bazenu največji pomen za normalno delovanje ribogojnice. Meritve se opravlja enkrat na minuto in se jih posreduje glavnemu opravilu preko MB7060Que vrste. Za branje podatkov iz senzorja smo uporabili kanal USART1 v povezavi s pinom PC11. Dodatna uporaba pina je bila potrebna, ker senzor namesto običajnega USART RX vhoda uporablja napetostno stanje, s katerim upravljamo delovanje senzorja. Poleg programske kode je bilo treba za delovanje senzorja dodati še inverter 4049. Inverter smo potrebovali, ker senzor za pošiljanje podatkov uporablja prilagojen protokol RS232, ki deluje na območju od 0 V do VCC. RS232 za razliko TTL protokola, ki se uporablja na ploščici STM, za začetni bit in pošiljanje podatkov uporablja ravno obratne logične vrednosti, zato smo morali uporabiti inverter, da smo signal popravili.

```
1
2  //opravilo za zajem in posiljanje podatkov gladine vode
3  void vTaskMB7060 (void *pvParameters){
4    //inicializacija spremenljivk
5     char depth[3];
6     int ind=1;
7    //izklop konstantnega merjenja podatkov
8     GPIO_ResetBits(GPIOC, GPIO_Pin_11);
9      while(1){
10       ind=1;
11       //1 minutna pavza
12         vTaskDelay(58000 / portTICK_RATE_MS);
13       //vklop senzorja
14         GPIO_SetBits(GPIOC, GPIO_Pin_11);
15       //cakanje na zajem podatkov
16         vTaskDelay(150 / portTICK_RATE_MS);
17       //izklop senzorja
18         GPIO_ResetBits(GPIOC, GPIO_Pin_11);
19       //priprava podatkov za posiljanje
20         while(txtMB7060[ind]!=0x0D){
21             depth[ind-1]=txtMB7060[ind];
22             ind++
23         }
24       //posiljanje podatkov v vrsto
25         xQueueSendToBack(MB7060Que,depth,portMAX_DELAY);
26     }
27 }
```

5.7 Pošiljanje podatkov na strežnik

vTaskControl je najpomembnejše opravilo v našem programu. Le-to skrbi za odvzem podatkov iz vseh senzorjev in jih pošilja naprej na strežnik. Zaradi glavne vloge v programu smo temu opravilu dodelili najvišjo prioriteto 1. Opravilo po inicializaciji spremenljivk vstopi v funkcijo configLan, ki skrbi za nastavitev IP-jev in ostalih pomembnih parametrov ethernet modula. Za tem pa vstopi v glavno zanko opravila, kjer prejema podatke, jih formatira ter nato pošlje naprej strežniku. Opravilo v glavni zanki pregleda vse vrste in si prekopira podatke, ki so na voljo. Nato iz pridobljenih podatkov formatira en string, ki se bo poslal na strežnik. Pred pošiljanjem podatkov resetiramo prekinitveni števec na 0 za lažje odčitavanje pravilnosti prenosa. Za tem sledi pošiljanje podatkov preko funkcije sendData. Po pošiljanju počakamo nekaj mikrosekund in nato preverimo status prenosa podatkov. V primeru, da je prišlo do napake prenosa, pošljemo podatke še enkrat. Za večkratno pošiljanje podatkov se nismo odločili, saj bi s tem samo obremenjevali linijo, napaka pa se najverjetneje ne bi odpravila. Po zaključenem prehodu zanke počakamo 30 sekund za nov obhod.

```
1
2  //glavno opravilo ki skrbi za formatiranje podatkov in posiljanje
       na streznik
3  void vTaskControl (void *pvParameters){
4    //inicializacija spremenljivk
5     int dhtData[2]={0,0};
6     char depthData[3];
7     char oxyData[3];
8     int waTempData[2]={0,0};
9     int ind;
10    portBASE_TYPE statusDHT;
11    portBASE_TYPE statusMB7060;
12    portBASE_TYPE statusDS18b20;
```

```
13   portBASE_TYPE statusDisOxy;
14   char allData[32];
15  //nastavitev ethernet modula
16   configLan();
17   vTaskDelay(1000 / portTICK_RATE_MS);
18  //vklop statusne ledice
19   GPIO_SetBits(GPIOD,GPIO_Pin_15);
20  // glavna zanka programa
21   while (1) {
22     //odvzem podatkov iz vrst
23      statusDHT = xQueueReceive(dht22Que,dhtData,0);
24       statusMB7060 = xQueueReceive(MB7060Que,depthData,0);
25      statusDS18b20= xQueueReceive(DS18b20Que,waTempData,0);
26      statusDisOxy= xQueueReceive(disOxyQue,oxyData,0);
27      vTaskDelay(500 / portTICK_RATE_MS);
28      ind=0;
29      //preveri kateri podatki so na voljo
30      if (statusMB7060 == pdTRUE || statusDS18b20 == pdTRUE ||
            statusDisOxy == pdTRUE || statusDHT == pdTRUE){
31           //formatiranje prejetih podatkov
32         if (statusDHT == pdTRUE){
33            formStringDHT(allData,dhtData,ind);
34         }
35         if (statusDS18b20 == pdTRUE){
36               formStringDS18B20(allData,waTempData,ind);
37         }
38         if (statusDisOxy == pdTRUE){
39               formStringDisOxy(allData,oxyData,ind);
40            }
41         if (statusMB7060 == pdTRUE){
42               formStringMB7060(allData,depthData,ind);
43         }
44         allData[ind]=0;
```

```
45          //posiljanje podatkov na streznik
46          countData5=0;
47          sendData(allData);
48          vTaskDelay(100 / portTICK_RATE_MS);
49          //preveri ce so bili podatki pravilno preneseni
50          if(txt[0]!='K'){
51            //v primeru napake ponovno posiljanje podatkov
52             GPIO_SetBits(GPIOD,GPIO_Pin_14);
53                vTaskDelay(1100 / portTICK_RATE_MS);
54                countData5=0;
55                sendData(allData);
56
57          }
58          else{
59             GPIO_ResetBits(GPIOD,GPIO_Pin_14);
60          }
61       }
62       vTaskDelay(30000 / portTICK_RATE_MS);
63    }
64 }
```

Poglavje 6

Načrtovanje in izdelava strežnika in spletne strani

Po vzpostavitvi vgrajenega sistema smo se lotili še izdelave strežnika (Slika 6.1). Glavna funkcija strežnika je pridobitev podatkov ter njihovo hranjenje. Za lažji dostop do podatkov smo naredili še enostavno spletno stran, ki prikazuje trenutno stanje na ribogojnici in omogoča prikaz podatkov, ki so bili v preteklosti zabeleženi. Strežniku smo dodali še dodatno funkcionalnost telefonskega klica za opozarjanje na padanje gladine vode ali vsebnosti kisika v bazenu. Za strežnik smo se odločili uporabiti Raspberry Pi 2, saj je cenovno najugodnejši in vsebuje vse funkcije, ki smo jih potrebovali. Za opravljanje klicev pa smo uporabili GSM modul Adafruit FONA, ki smo ga z Raspberry-em povezali preko UART protokola. Na 32GB microSD kartico smo s pomočjo programa Win32 Disk Imager namestili operacijski sistem Raspberian. Po vzpostavitvi operacijskega sistema je bilo treba namestiti še strežnik Apache, MySql-server, PHP5, MySql WorkBench, Python 2.7 in RRDtool.

Spletno stran se je razvijalo na računalniku z operacijskim sistemom Windows 8.1. Za pisanje spletne strani smo uporabili program WebMatrix, za testiranje pa smo naložili brezplačni WAMP strežnik.

Slika 6.1: Slika našega Raspberry Pi strežnika.

6.1 Mysql baza podatkov

Za hranjenje podatkov smo uporabili MySql. Najprej je bilo treba ustvariti bazo, ki smo jo poimenovali ribogojnicadb. Za tem smo ustvarili 6 tabel, ki zaradi načina beleženja nimajo relacij. Za lažjo predstavitev izgleda baze smo spodaj vključili shemo, zajeto iz programa MySql Workbench (Slika 6.2). Štiri izmed šestih tabel imajo funkcijo beleženja podatkov. Te table so flow, humidity, temperature in watertemperature. Kot lahko opazite, nimamo tabele z globino vode in tabele z vsebnostjo kisika. Teh tabel nismo ustvarili iz razloga, ker so ti podatki pomembni le za prikaz trenutnega stanja in ni potrebe po dolgotrajnem hranjenju. Vse štiri zgoraj navedene table imajo enako zgradbo, in sicer vsebujejo stolpec z imenom dateTime, ki vsebuje datum in čas zabeleženega podatka, hkrati pa služi kot primarni ključ. Poleg

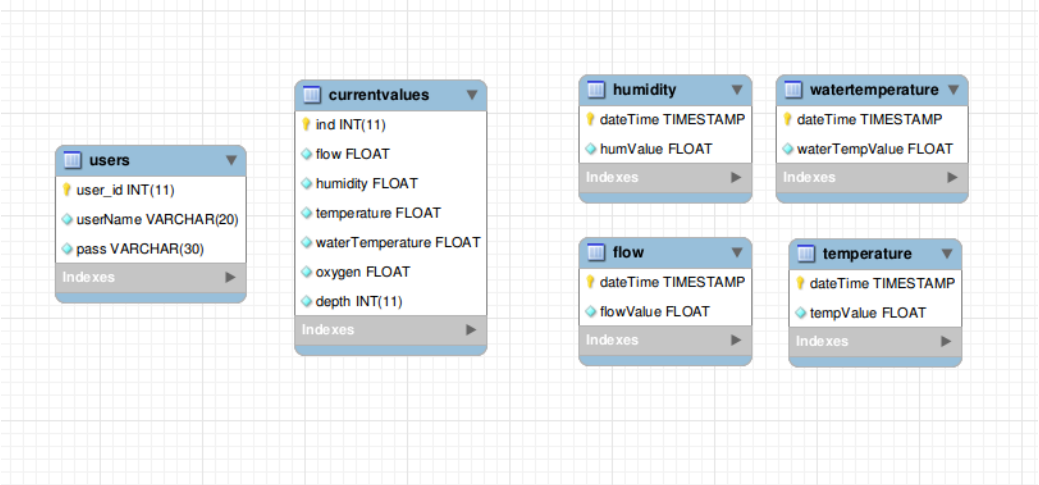
časovnega stolpca ima vsaka tabela še svoj stolpec za podatke, ki smo ga glede na tabelo različno poimenovali.

Preostali dve kreirani tabeli pa služita kot podpora spletni strani. Tabela users je namenjena hranjenju uporabnikov za prijavo na spletno stran. Tabela vsebuje tri stolpce z imeni:

- ind
- userName
- pass

Druga tabela z imenom currentvalues pa služi prikazu trenutnih podatkov v ribogojnici in se s časom prepisuje. Vsebuje 6 stolpcev z naslednjimi stolpci:

- ind
- flow
- humidity
- temperature
- waterTemperature
- oxygen
- depth



Slika 6.2: Shema podatkovna baze ribogojnicadb.

6.2 RRD baza podatkov

Za kratkotrajno hranjenje podatkov in izris grafov smo uporabili RRD bazo. Za beleženje vseh podatkov je bilo treba narediti šest ločenih baz. Bazi, namenjeni hranjenju podatkov o vsebnosti kisika in globini vode, sta zelo kratkoročni, saj lahko prikazujeta le razpon enega meseca, medtem ko baze, ki so namenjene hranjenju temperaturnih vrednosti, vlage, pretoka in temperature vode, lahko prikažejo povprečne vrednosti za interval 10 let, vendar z nižjo natančnostjo. Opisali bomo le eno iz vsake skupine, saj se od ostalih razlikujejo le po imenu.

Baza, namenjena hranjenju povprečnih vrednosti globine vode, se začne s številko 1300000000, kar predstavlja število sekund od 1. 1. 1970 0:00. V bazo bodo podatki prihajali vsakih 60 sekund in ker so to vrednostni podatki in ne števci, uporabimo spremenljivko tipa GAUGE. Če podatek ne pride v intervalu 60 sekund, smo dodali še 30-sekundni varnostni interval in s tem dobili 90-sekundni utrip (*ang. heartbeat*). Bazi smo dodali še minimalno in maksimalno vrednost za preverjanje pravilnosti podatkov. Za hranjenje podatkov smo ustvarili 4 arhive, ki hranijo povprečne vrednosti 1 ure, 1 dneva,

1 tedna in 1 meseca. Vsi ustvarjeni arhivi zahtevajo za izračun povprečne vrednosti vsaj 90 % prejetih podatkov. Za arhiv, ki prikazuje interval ene ure, smo uporabili 1-minutno natančnost prikaza podatkov. Za dnevni arhiv smo uporabili 10-minutno natančnost, za tedenski urno (60 min), za mesečni pa enodnevno (1440 min) natančnost.

Za izračun časovnega intervala je treba zmnožiti korak (*ang. step*) s številom korakov, iz katerih želimo izračunati povprečno vrednost. Izračunan zmnožek moramo še pomnožiti s številom povprečij, ki jih hranimo v bazi.

Primer: [60 (step=1min) * 10 (natančnost=10min)] * 144 (število povprečji v bazi) = 86400 s = 1 dan

```
rrdtool create depth.rrd \
 --start 1300000000 \
 --step 60 \
 DS:depth:GAUGE:90:0:800 \
 # 1m --> 1h
 RRA:AVERAGE:0.9:1:60 \
 #10m --> 1d
 RRA:AVERAGE:0.9:10:144 \
 # 1h --> 1w
 RRA:AVERAGE:0.9:60:168 \
 # 1d --> 1mes
 RRA:AVERAGE:0.9:1440:30
```

Za kreiranje baze z 10-letnim razponom je postopek enak, le da se uporabi druge parametre. Pri bazi z večjimi časovnimi razponi je treba biti pozoren na parameter start, kajti v nasprotnem primeru pride do napake izrisovanja grafov, ker se zahteva podatke, ki so pred teoretičnim nastankom baze. Zato smo mi uporabili število 1000000000, ki predstavlja konec leta 2001. Ker podatke iz senzorjev pobiramo enkrat na uro, je tak tudi naš korak. Tudi v tej bazi smo uporabili štiri arhive, in sicer za dnevni, mesečni, letni in 10-letni prikaz povprečij. Pri dnevnem intervalu imamo enourno natančnost,

pri mesečnem smo uporabili dnevno, pri letnem tedensko in pri 10-letnem mesečno natančnost.

```
rrdtool create flow.rrd \
 --start 1000000000 \
 --step 3600 \
 DS:flow:GAUGE:4000:0:200 \
 # 1h --> 24h
 RRA:AVERAGE:0.9:1:24 \
 # 1d --> 1mes
 RRA:AVERAGE:0.9:24:30 \
 # 1w --> 1y
 RRA:AVERAGE:0.9:168:52 \
 # 1mes --> 10y
 RRA:AVERAGE:0.9:720:120
```

6.3 Strežnik za pridobivanje podatkov

Za pridobivanje podatkov od STM smo napisali svoj strežnik v programskem jeziku Python 2.7 . Za normalno delovanje programa je bilo treba dodati pet knjižnic:

- socket
- datetime
- sys
- rrdtool
- MySQLdb
- os
- time
- wiringPi2

Na začetku programa smo najprej odprli vtič (*ang. socket*) za lastnim IP naslovom in prehodom (*ang. port*) 8234. Za tem je sledila glavna zanka programa, ki je preverjala prihod podatkov in njihovo pravilnost. Po preverjanju podatkov se pridobi trenutni čas sistema in se skupaj s podatki shrani v obe bazi. Če so se podatki pravilno prenesli in shranili v MySql in RRD bazo, je program STM ploščici odgovoril z znakom K. V primeru kakršnekoli napake je strežniški program STM-u vrnil znak 'E'. Če smo med preverjanjem zaznali, da je globina vode oziroma vsebnost kisika v vodi prenizka, se preko GSM modula najprej pošlje SMS opozorilo s stanjem sistema, za tem se kot dodatni varovalni ukrep opravi še klic, saj ga je težje preslišati kot SMS obvestilo.

```
1
2  os.chdir("/var/www")
3  server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4  server.bind(("192.168.1.111", 8234))
5  server.listen(1)
6  con = mdb.connect(host='localhost', user='root', passwd='riba2',
       db='ribogojnicadb')
7  db = con.cursor()
8  while 1:
9     # povezovanje s posiljateljem
10     conn, clientAddr = server.accept()
11     print("Connection address:", clientAddr)
12     while 1:
13         errStr = "K\n"
14         # prejemanje podatkov
15         data = conn.recv(BUF_SIZE)
16         if not data:
17             break
18         dataList = data.split(",")
19         for i in range(0, len(dataList)):
20                 measure = dataList[i][0]
```

```
21              valueString = dataList[i][1:]
22        try:
23              if measure == "H":
24                    valueH = float(valueString)
25                 sqlData = "INSERT INTO humidity VALUES(NOW(),%f);"
                       % valueH
26                 sqlData2 = "UPDATE currentvalues SET ind=0,
                       humidity=%f;" % valueH
27                 ret=rrdtool.update('hum.rrd', 'N:%f' %valueH)
28                 db.execute(sqlData)
29                 db.execute(sqlData2)
30              elif measure == "T":
31                  valueT = float(valueString)
32                 sqlData = "INSERT INTO temperature
                       VALUES(NOW(),%f);" % valueT
33                 sqlData2 = "UPDATE currentvalues SET ind=0,
                       temperature=%f;" % valueT
34                 ret=rrdtool.update('temp.rrd', 'N:%f' %valueT)
35                    db.execute(sqlData)
36                 db.execute(sqlData2)
37              elif measure == "W":
38                 valueW = float(valueString)
39                 sqlData = "INSERT INTO watertemperature
                       VALUES(NOW(),%f);" % valueW
40                 sqlData2 = "UPDATE currentvalues SET ind=0,
                       waterTemperature=%f;" % valueW
41                 ret=rrdtool.update('watertemp.rrd', 'N:%f' %valueW)
42                 db.execute(sqlData)
43                 db.execute(sqlData2)
44           elif measure == "O":
45                  valueO = float(valueString)
46                 if valueO < 3:
47                    alert(valueO,"Kisik")
```

```
48                  sqlData2 = "UPDATE currentvalues SET ind=0,
                        oxygen=%f;" % valueO
49                  ret=rrdtool.update('oxygen.rrd', 'N:%f' %valueO)
50                  db.execute(sqlData2)
51              elif measure == "D":
52                  valueD = int(valueString)
53                  if valueD < 200:
54                     alert(valueD,"Globina")
55                  sqlData2 = "UPDATE currentvalues SET ind=0,
                        depth=%d;" % valueD
56                  ret=rrdtool.update('depth.rrd', 'N:%f' %valueD)
57                  db.execute(sqlData2)
58              con.commit()
59              conn.send(errStr.encode('utf-8'))
60          except: #...
61  conn.close()
```

6.4 Vzpostavitev modula Adafruit FONA in pošiljanje SMS-ov ter klicanje

Za obveščanje v primeru kritičnega stanja smo uporabili modul Adafruit FONA. Na Raspberry smo ga povezali preko pinov 14 in 15, ki so fiksno dodeljeni UART komunikaciji, in pina 18, ki smo ga uporabili za reset modula. Preden smo se lotili pisanja programske kode, je bilo treba popraviti datoteko /boot/cmdline.txt, saj Raspberry primarno uporablja UART za simulacijo ukazne vrstice. Po izklopu primarnih nastavitev smo v python-u napisali funkcijo alert, ki skrbi za inicializacijo, vklop modula, opravljanje klicev in pošiljanje SMS-ov. Na začetku smo morali najprej inicializirati pine s pomočjo knjižnice wiringpi2. Nato je bilo treba reseterirati modul in počakati, da se ponovno vklopi. Za inicializacijo povezave je treba modulu trikrat poslati ukaz AT, na kar nam odgovori z OK. Po inicializaciji pove-

zave smo s petimi ukazi preverili, ali se je modul uspešno povezal v omrežje. Na koncu imamo še dve funkciji, ki skrbita za izvršitev potrebnih ukazov za pošiljanje SMS-ov in klicanje.

```
1
2  def alert(value,dataType):
3    # inicializacija UART-a in pina 18
4     wiringpi.wiringPiSetup()
5     fd = wiringpi.serialOpen('/dev/ttyAMA0',9600)
6     wiringpi.pinMode(1,1)
7    # reset modula pred delovanje
8     wiringpi.digitalWrite(1,1)
9     time.sleep(0.01)
10    wiringpi.digitalWrite(1,0)
11    time.sleep(0.1)
12    wiringpi.digitalWrite(1,1)
13    time.sleep(3)
14   # inicializacija povezave
15    sendWaitReplay("AT\r","OK\r",fd)
16    time.sleep(0.1)
17    sendWaitReplay("AT\r","OK\r",fd)
18    time.sleep(0.1)
19    sendWaitReplay("AT\r","OK\r",fd)
20    time.sleep(0.1)
21   # izklop odmeva (ECHO)
22    sendWaitReplay("ATE0\r","OK\r",fd)
23    time.sleep(0.1)
24   # preverjanje delovanja
25    ATcommResp("AT+CREG?\r",fd)
26    time.sleep(0.1)
27   # preverjanje signala
28    ATcommResp("AT+CSQ\r",fd)
29    time.sleep(0.1)
30   # preverjanje ponudnika
```

```
31   ATcommResp("AT+COPS?\r",fd)
32   time.sleep(0.1)
33  # vklop posiljanja SMS-ov
34   ATcommResp("AT+CMGF=1\r",fd)
35    time.sleep(0.1)
36  # preverjanje nastavitve
37    ATcommResp("AT+CMGF?\r",fd)
38    time.sleep(0.1)
39  # posiljanje sms z ukazom AT+CMGS ki potrebuje <CR>(\r) in
        CTRL+Z(0x1A) za konec
40   sendSMS("%s je pod normalno vrednostjo !!! %f\r\x1a"
        %(dataType,value),number,fd)
41  # klic stevilke
42   voiceCall(number,fd)
43   return
```

6.5 PHP spletna stran

Za prikaz podatkov uporabniku smo izdelali spletno stran. Pri izbiri tehnologije smo se odločili za PHP5, ker edini podpira RRDtool, ki ga potrebujemo za izrisovanje grafov. Spletno stran smo napisali s pomočjo programa WebMatrix3, ki je namenjen razvoju HTML spletnih strani. Za oblikovanje spletne strani smo uporabili CSS, ki nam je omogočil enostavno in enotno oblikovanje. Za risanje grafov in pridobivanje podatkov iz baze smo uporabili PHP 5. Zaradi uporabe PHP je bilo treba namestiti WAMP za razvoj spletne strani in LAMP za prikaz spletne strani preko Raspberry Pi. Pred vstopom na domačo stran smo se odločili za uporabo prijave in s tem zagotovili vsaj minimalno varnost. Na prvi strani (Slika 6.3) se nahajata samo polji za vnos uporabniškega imena in gesla ter gumb za prijavo. Ob pritisku na gumb se izvede koda, ki pregleda, ali je uporabnik v bazi ter nastavi parametre za začetek seje. Na začetku vsake strani smo vključili še dodaten del kode, ki

preverja, če je seja vzpostavljena, v nasprotnem primeru pa nas preusmeri na stran za prijavo. Poleg seje smo uporabili še sejni piškotek (*ang. session cookie*), ki nam onemogoči ohranitev seje po zaprtju brskalnika.

Slika 6.3: Prijavna stran.

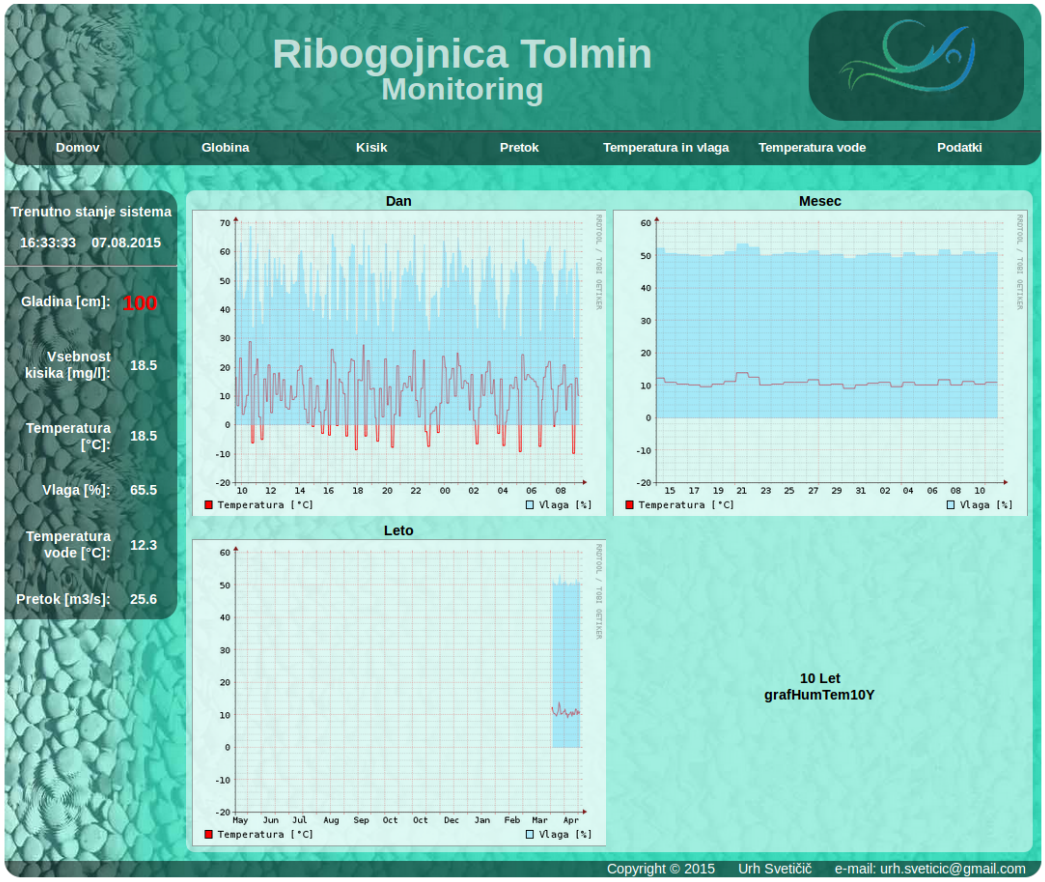
Po uspešni prijavi se nam odpre domača stran. Na njej lahko v levem kotu najdemo podatke, ki prikazujejo trenutno stanje v ribogojnici, na sredini pa imamo grafe z najkrajšim časovnim intervalom. Če si želimo pogledati širši časovni interval, je treba izbrati zavihek z imenom iskane spremenljivke. Ob odprtju zaželene strani se nam prikaže podobna stran, kot jo lahko vidimo na spodnji sliki (Slika 6.4). Razpored je podoben kot na domači strani; v levem kotu imamo trenutno stanje sistema, na sredini pa se nahajajo grafi. Za vsako spremenljivko imamo narejene štiri grafe z različnimi časovnimi intervali. Za izris grafov smo uporabili RRDtool, primer kode za izris grafa se nahaja spodaj.

Desetletni graf na spodnji sliki (Slika 6.4) ni prikazan, ker se je uporabila le testna baza, v kateri pa ni bilo dovolj podatkov, da bi lahko prikazali

takšen časovni razpon.

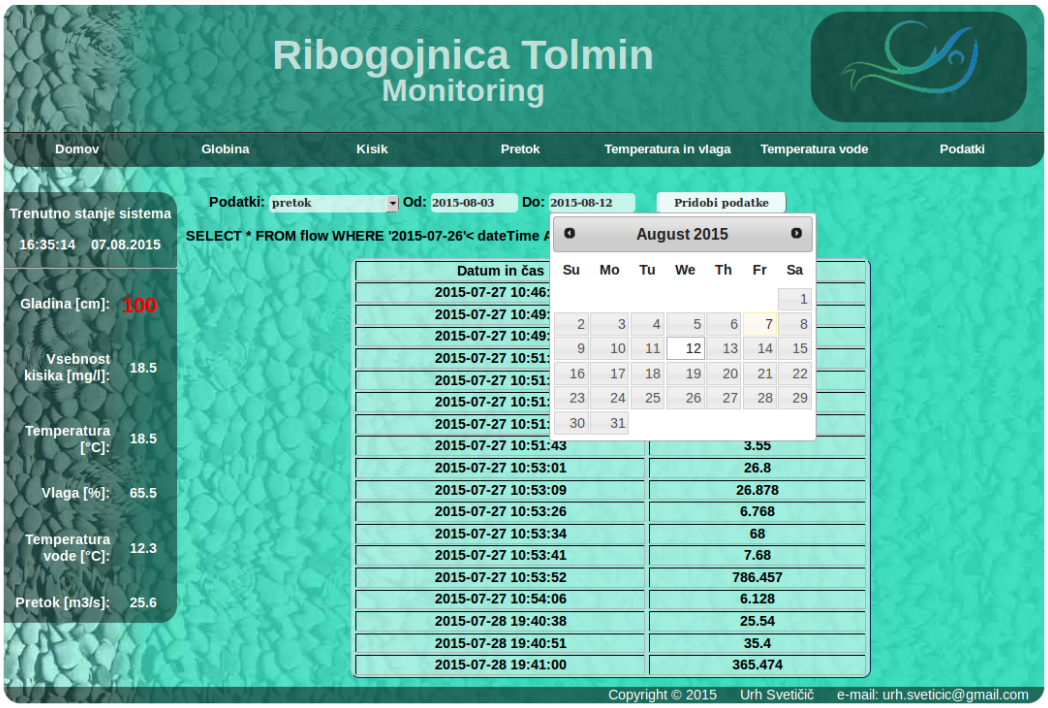
```
1
2   $options = array(
3      "--slope-mode",
4      "--start", "N-2505600",
5      "--end", "N",
6      "-h","300",
7      "-l","-20",
8      "--x-grid", "DAY:1:DAY:7:DAY:2:86400:%d",
9      "--color","BACK#FFFFFFA0",
10     "--color","CANVAS#FFFFFF00",
11     "--color","SHADEA#00000060",
12     "--color","SHADEB#00000060",
13     "--border","1",
14     "DEF:temps=task.rrd:task:AVERAGE",
15     "DEF:hums=task.rrd:task:AVERAGE",
16     "LINE:temps#FF0000:Temperatura [C]",
17     "AREA:hums#70DBFF80:Vlaga [%]",
18  );
19  $ret = rrd_graph("img/graphHumTemD.png", $options);
```

Ker se iz grafov ne da razbrati natančne vrednosti in se z večanjem časovnega intervala zgublja natančnost zaradi uporabe povprečnih vrednosti, smo se odločili uporabiti še bazo MySql. Ta bo omogočala prikaz natančnih podatkov tudi za več let nazaj. Za dostop do teh podatkov smo naredili stran, pri čemer je treba na začetku izbrati bazo, iz katere želimo pridobiti podatke, nato pa je treba navesti še časovni razpon za prikaz podatkov. Za boljšo uporabniško izkušnjo smo za izbiro baze uporabili izvlečni seznam (*ang. dropdown menu*), za določitev datuma pa smo implementirali maj-



Slika 6.4: Grafični prikaz podatkov.

hen koledarček, ki se prikaže ob kliku na izbrano polje. Spletna stran za pridobitev podatkov iz baze je prikazana na spodnji sliki (Slika 6.5).



Slika 6.5: Pridobivanje podatkov iz baze.

Poglavje 7

Sklepne ugotovitve

V diplomskem delu smo razvili celovit sistem nadzora ribogojnice. Sistem pridobiva podatke iz štirih senzorjev, ki so na STM32F4 Discovery priklopljeni preko različnih komunikacijskih protokolov (U(S)ART, 1-wire in serial). Razvojna ploščica STM32F4 jih nato ponovno preko UART protokola pošlje na ethernet modul. Podatki se iz modula preko protokola TCP/IP prenesejo na Raspberry Pi strežnik, kjer se shranijo v MySql in RRD bazo. Za oddaljeni nadzor nad ribogojnico smo naredili spletno stran, ki je zaščitena z imenom in geslom. Poleg nadziranja stanja preko spletne strani smo implementirali še opozarjanje z SMS sporočili in klicem. Za obveščanje smo uporabili GSM modul, ki je bil priklopljen na Raspberry Pi preko U(S)ART vmesnika. Med izdelavo sistema smo prišli tudi do nekaj možnih izboljšav. Največja izboljšava bi bila namestitev kamere, kar bi nam omogočalo prikaz dejanskega stanja na ribogojnici. Z dodelavo te izboljšave bi odpravili največjo pomanjkljivost našega sistema. Druga možna izboljšava bi bila namestitev globinomerskih senzorjev na vse bazene. Vendar bi morali v tem primeru popraviti strežniški program in mu dodati niti, saj je trenutno sposoben ohranjati le eno povezavo.

Z diplomsko nalogo sem pridobil veliko znanja o razvojni ploščici STM32F4 Discovery ter senzorjih. Naučil sem se programskega jezika Python, ki sem ga do izdelave diplomske naloge le nekajkrat uporabil. Poleg tega sem se naučil še izdelovati spletne strani in uporabljati MySql, RRDtool in PHP. Sistem trenutno deluje v novi ribogojnici na Tolminki. Načrtuje pa se izdelavo okrnjene verzije za starejšo ribogojnico.

Slike

Literatura

[1] Woynarovich, András. and Hoitsy, György. and Moth-Poulsen, Thomas. and Food and Agriculture Organization of the United Nations, "Small-scale rainbow trout farming", Food and Agriculture Organization of the United Nations Rome, 2011, http://www.fao.org/docrep/015/i2125e/i2125e.pdf, [Dostopano 30. 7. 2015].

[2] B. Skalin, Ribogojstvo, *Kmečki glas*, http://books.google.si/books?id=cxzkAAAACAAJ,1993.

[3] R. Kamnik, Robotski in merilni vgrajeni sistemi, *Založba FE in FRI*, http://robo.fe.uni-lj.si/ kamnikr/sola/RiMVS/

[4] Serial and UART. [Online]. Dosegljivo: https://www.freebsd.org/doc/en_US.ISO8859-1/articles/serial-uart/. [Dostopano 30. 7. 2015].

[5] Shema UART. [Online]. Dosegljivo: http://www.socialledge.com/sjsu/index.php?title= File:S15_Cmpe146_Coffee_Alarm_UartFrame.png. [Dostopano 30. 7. 2015].

[6] Spletna dokumentacija razvojne ploščice STM32F407VG. [Online]. Dosegljivo: http://www.st.com/web/catalog/mmc/FM141/SC1169/SS1577/LN11/PF252140. [Dostopano 30. 7. 2015].

[7] Shema GPIO. [Online]. Dosegljivo: http://cfile3.uf.tistory.com/image/177800434EDC52D8169342. [Dostopano 30. 7. 2015].

[8] Opis operacijskega sistema FreeRTOS. [Online]. Dosegljivo: http://www.freertos.org/. [Dostopano 30. 7. 2015].

[9] Shema prehajanj med opravili. [Online]. Dosegljivo: http://www.freertos.org/tskstate.gif. [Dostopano 30. 7. 2015].

[10] Opis programskega okolja IAR Embedded Workbench na spletnem mestu Wikipedia. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/IAR_Systems. [Dostopano 30. 7. 2015].

[11] Spletna dokumentacija senzorja DHT22. [Online]. Dosegljivo: http://www.adafruit.com/datasheets/DHT22.pdf . [Dostopano 30. 7. 2015].

[12] Spletna dokumentacija ethernet modula USR-TCP232-T. [Online]. Dosegljivo: http://www.tcp232.net/download/USR-TCP232-T-EN.pdf. [Dostopano 30. 7. 2015].

[13] Opis senzorja MB7060. [Online]. Dosegljivo: http://www.maxbotix.com/Ultrasonic_Sensors/MB7060.htm. [Dostopano 30. 7. 2015].

[14] Spletna dokumentacija temperaturnega senzorja DS18B20. [Online]. Dosegljivo: http://www.adafruit.com/datasheets/DS18B20.pdf. [Dostopano 30. 7. 2015].

[15] Opis in dokumentacija Atlas EZO D.O. senzorja na uradni spletni strani. [Online]. Dosegljivo: http://www.atlas-scientific.com/product_pages/kits/do_kit.html?. [Dostopano 30. 7. 2015].

[16] Opis Raspberry Pi na uradni spletni strani. [Online]. Dosegljivo: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/. [Dostopano 30. 7. 2015].

[17] Opis in prikaz uporabe GSM modula Adafruit FONA na uradni spletni strani. [Online]. Dosegljivo: https://learn.adafruit.com/adafruit-fona-mini-gsm-gprs-cellular-phone-module/overview. [Dostopano 30. 7. 2015].

[18] Opis spletnega strežnika Apache. [Online]. Dosegljivo: http://httpd.apache.org/ABOUT_APACHE.html. [Dostopano 30. 7. 2015].

[19] Opis programskega jezika PHP na uradni spletni strani. [Online]. Dosegljivo: http://php.net/manual/en/intro-whatis.php. [Dostopano 30. 7. 2015].

[20] Opis programskega jezika PHP na spletnem mestu Wikipedia. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/PHP. [Dostopano 30. 7. 2015].

[21] Spletna dokumentacija z uporabo za podatkovno bazo RRDtool. [Online]. Dosegljivo: http://oss.oetiker.ch/rrdtool/tut/rrdtutorial.en.html. [Dostopano 30. 7. 2015].

[22] Opis relacijske podatkovne baze MySql na spletnem mestu Wikipedia. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/MySQL. [Dostopano 30. 7. 2015].

[23] Opis razvojnega orodja za relacijske podatkovne baze MySql na spletnem mestu Wikipedia. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/MySQL_Workbench. [Dostopano 30. 7. 2015].

[24] Opis skriptnega jezika HTML na spletnem mestu w3schools. [Online]. Dosegljivo:
http://www.w3schools.com/html/. [Dostopano 30. 7. 2015].

[25] Opis skriptnega jezika HTML na spletnem mestu Wikipedia. [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/HTML. [Dostopano 30. 7. 2015].

[26] Opis CSS na spletnem mestu Wikipedia. [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Cascadingq_Style_Sheets. [Dostopano 30. 7. 2015].

[27] Opis in prikaz uporabe CSS na spletnem mestu w3schools. [Online]. Dosegljivo:
http://www.w3schools.com/css/. [Dostopano 30. 7. 2015].